A range minima parallel algorithm for coarse grained multicomputers*

H. Mongelli[1] and S. W. Song2

[1] Universidade Federal do Mato Grosso do Sul
and Universidade de São Paulo <mongelli at ime. usp. br>
[2] Universidade de São Paulo <song at ime. usp. br>

Abstract. Given an array of n real numbers $A = (a_1, a_2, \dots, a_n)$, define $MIN(i,j) = \min\{a_i, \dots, a_j\}$. The range minima problem consists of preprocessing array A such that queries $MIN(i,j)$, for any $1 \le i \le j \le n$, can be answered in constant time. Range minima is a basic problem that appears in many other important problems such as lowest common ancestor, Euler tour, pattern matching with scaling, etc. In this work we present a parallel algorithm under the CGM model (Coarse Grained Multicomputer), that solves the range minima problem in $O(\frac{n}{p})$ time and constant number of communication rounds.

1 Introduction

Given an array of n real numbers $A = (a_1, a_2, \dots, a_n)$, define $MIN(i,j) = \min\{a_i, \dots, a_j\}$. The range minima problem consists of preprocessing array A such that queries $MIN(i,j)$, for any $1 \le i \le j \le n$, can be answered in constant time.

The parallel computing model used in this work is the CGM (Coarse Grained Multicomputer). The CGM(n,p) consists of an input of size $O(n)$ and p processors with point-to-point communication. An algorithm under this model consists of supersteps that involve a local computing round and a communication round between processors. In a communication round, each processor can send and/or receive $O(\frac{n}{p})$ data.

This is a more realistic model than the PRAM model for most real parallel computers. The number of communication phases is taken into consideration in the design of algorithms. It has been introduced in [7, 8] where algorithms using a constant number of communication rounds were presented for several Computational Geometry problems. Algorithms requiring $O(\log p)$ communication rounds have been presented for the solution of several graph problems[6].

In this work we consider the range minima problem. We present an algorithm under the CGM(n,p) model, that solves the problem requiring a constant number of communication rounds and $O(\frac{n}{p})$ computation time. In the PRAM model, there is an optimal algorithm of $O(\log n)$ time [12]. This algorithm, however, is

* Supported by FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) - Proc. No. 98/06138-2, and CNPq - Proc. No. 52 3778/96-1 and CAPES.

not immediately transformable to the CGM model. Berkman et al. [4] describe an optimal algorithm of $O(\log\log n)$ time for a PRAM CRCW. This algorithm motivates the CGM algorithm presented in this paper.

Range minima is a basic problem and appears in other problems. In [12], range minima is used in the design of a PRAM algorithm for the problem of Lowest Common Ancestor (LCA). This algorithm uses twice the Euler-tour problem. The idea of this algorithm can be used in the CGM model, by using the same sequence of steps. In [6], the Euler-tour problem in the CGM model is discussed. Its implementation reduces to the list ranking problem, that uses $O(\log p)$ communication rounds in the CGM model [6]. Applications of the LCA problem, on the other hand, are found in graph problems (see [15]) and in the solution of some other problems. Besides the LCA problem, the range minima problem is used in several other applications [3, 10, 14].

2 The CGM model and some definitions

The CGM model was proposed by Dehne [7, 8]. It is similar to the BSP (Bulk-Synchronous Parallel) model[17]. However, it uses only two parameters: the input size n and the number of processors p, each one with a local memory of size $O(\frac{n}{p})$. The term *coarse grained* means the size of the local memory is much larger than $O(1)$. Each processor is connected by a router that can send messages in a point-to-point fashion. A CGM algorithm consists of an alternate sequence of computing and communication phases separated by a barrier synchronization. A computing phase is equivalent to a superstep in the BSP model. In a computing phase, we usually use the best sequential algorithm in each processor to process its data locally.

In each communication phase or round, each processor exchanges a maximum of $O(\frac{n}{p})$ data with other processors. In the CGM model, the communication cost is modeled by the number of communication phases. Some algorithms for Computational Geometry and graph problems require a constant number or $O(\log p)$ communication rounds [6–9]. Contrary to a PRAM algorithm, that is often designed for $p = O(n^k)$, with $k \in \mathcal{N}$ and each processor receives a small number of input data, in this model we consider the more realistic cases where $n \gg p$. The CGM model is particularly suitable in current parallel machines in which the global computing speed is considerably larger than the global communication speed.

Definition 1 *Consider an array $A = (a_1, a_2, \ldots, a_n)$ of n real numbers. The **prefix minimum** array is the array $P = (\min\{a_1\}, \min\{a_1, a_2\}, \ldots, \min\{a_1, a_2, \ldots, a_n\})$. The **suffix minimum** array is the array $S = (\min\{a_1, a_2, \ldots, a_n\}, \min\{a_2, \ldots, a_n\}, \ldots, \min\{a_{n-1}, a_n\}, \min\{a_n\})$.*

Definition 2 *The **lowest common ancestor** of two vertices u and v of a rooted tree is the vertex w that is an ancestor of u and v and that is farthest from the root. We denote $w = LCA(u, v)$.*

Definition 3 *The* ***lowest common ancestor problem*** *consists of preprocessing a rooted tree such that queries of the type $LCA(u, v)$, for any vertices u and v of the tree, can be answered in constant sequential time.*

3 Sequential algorithms

The algorithm for the range minima problem, in the CGM model, uses two sequential algorithms that are executed using local data in each processor. These algorithms are described in the following.

3.1 Algorithm of Gabow et al.

This sequential algorithm was designed by Gabow et al. [10] and uses the **Cartesian tree** data structure, introduced in [18]. An example of Cartesian tree is given in figure 1. In this algorithm we need also an algorithm for the lowest common ancestor problem.

Definition 4 *Given an array $A = (a_1, a_2, \ldots, a_n)$ of n distinct real numbers, the* ***Cartesian tree*** *of A is a binary tree whose nodes have as label the values of array A. The root of the tree has as label $a_m = \min\{a_1, a_2, \ldots, a_n\}$. Its left subtree is a Cartesian tree for $A_{1,m-1} = (a_1, a_2, \ldots, a_{m-1})$, and its right subtree is a Cartesian tree for $A_{m+1,n} = (a_{m+1}, \ldots, a_n)$. The Cartesian tree for an empty array is the empty tree.*

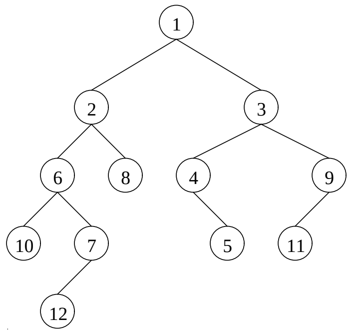

Fig. 1. Cartesian tree corresponding to the array $(10, 6, 12, 7, 2, 8, 1, 4, 5, 3, 11, 9)$.

ALGORITHM 1: Range Minima (Gabow et al.)

1. Build a Cartesian tree for A.
2. Use a linear sequential algorithm for the LCA problem using the Cartesian tree.

The construction of the Cartesian tree takes linear time [10]. Linear sequential algorithms for the LCA problem can be found in [5, 11, 16]. Thus, any query $MIN(i, j)$ is answered as follows. From the recursive definition of the Cartesian tree, the value of $MIN(i, j)$ is the value of the LCA of a_i and a_j. Thus, each range minima query can be answered in constant time through a query of LCA in the Cartesian tree. Therefore, the range minima problem is solved in linear time.

3.2 Sequential algorithm of Alon and Schieber

In this section we describe the algorithm of Alon and Schieber [2] of $O(n \log n)$ time. In spite of its non linear complexity, this algorithm is crucial in the description of the CGM algorithm, as will be seen in section 4. Without loss of generality, we consider n to be a power of 2.

ALGORITHM 2: Range Minima (Alon and Schieber)

1. Construct a complete binary tree T with n leaves.
2. Associate the elements of A to the leaves of T.
3. For each vertex v of T we compute the arrays P_v and S_v, the prefix minimum and the suffix minimum arrays, respectively, of the elements of the leaves of the subtree with root v.

Tree T constructed by algorithm 2 will be called **PS-tree**. Figure 2 illustrates the execution of this algorithm. Queries of type $MIN(i, j)$ are answered as follows. To determine $MIN(i, j)$, $1 \leq i \leq j \leq n$, we find $w = \text{LCA}(a_i, a_j)$ in T. Let v and u be the left and right sons of w, respectively. Then, $MIN(i, j)$ is the minimum between the value of S_v in the position corresponding to a_i and the value of P_u in the position corresponding to a_j. PS-tree is a complete binary tree. In [12, 15] it is shown that a LCA query in complete binary trees can be answered in constant time.

4 The CGM algorithm for the range minima problem

From the sequential algorithms seen in the previous section, we can design a CGM algorithm for the range minima problem. This algorithm is executed in $O(\frac{n}{p})$ time and uses $O(1)$ communication rounds. The major difficulty is how to store the required data structure among the processors so that the queries can be done in constant time, without violating the limit of $O(\frac{n}{p})$ memory, required by the CGM model.

Given an array $A = (a_1, a_2, \ldots, a_n)$, we write $A[i] = a_i$, $1 \leq i \leq n$, and $A[i \ldots j] =$ subarray $(a_i, \ldots, a_j)$, $1 \leq i \leq j \leq n$.

The idea of the algorithm is based on how queries $MIN(i, j)$ can be answered. Each processor stores $\frac{n}{p}$ contiguous positions of the input array. Thus, given $A = (a_1, a_2, \ldots, a_n)$, we have subarrays $A_i = (a_{i\frac{n}{p}+1}, \ldots, a_{(i+1)\frac{n}{p}})$, for $0 \leq i \leq p - 1$. Depending on the location of a_i and a_j in the processors, we have the following cases:

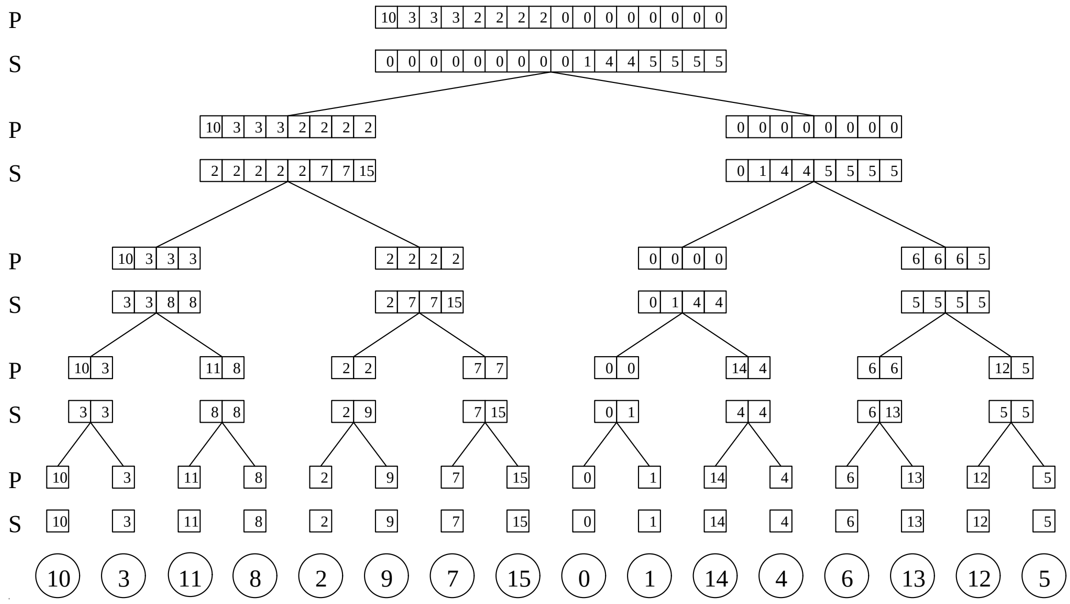

Fig. 2. PS-tree generated by the algorithm of Alon and Schieber for the array $(10, 3, 11, 8, 2, 9, 7, 15, 0, 1, 14, 4, 6, 13, 12, 5)$.

1. if a_i and a_j are in a same processor, the problem domain reduces to the subarray stored in that processor. Thus, we need a data structure to answer this type of queries in constant time. This data structure is obtained in each processor by Algorithm 1 (section 3.1).
2. if a_i and a_j are in distinct processors $p_{\bar{i}}$ e $p_{\bar{j}}$ (without loss of generality, $\bar{i} < \bar{j}$), respectively, we have two subcases:

   (a) if $\bar{i} = \bar{j} - 1$, a_i and a_j are in neighbor processors, $MIN(i, j)$ corresponds to the minimum between the minimum of a_i through the end of array $A_{\bar{i}}$ and the minimum of the beginning of $A_{\bar{j}}$ to a_j. These minima can be determined by Algorithm 1. To determine the minimum of the minima we need one communication round.

   (b) if $\bar{i} < \bar{j} - 1$, $MIN(i, j)$ corresponds to the minimum among the minimum of subarray $A_{\bar{i}}[i - \bar{i}\frac{n}{p} \dots (i+1)\frac{n}{p}]$, the minimum of subarray $A_{\bar{j}}[\bar{j}\frac{n}{p} + 1 \dots j - \bar{j}\frac{n}{p}]$ and the minima of subarrays $A_{\bar{i}+1}, \dots, A_{\bar{j}-1}$. The first two minima are obtained as in the previous subcase. The minima of subarrays $A_{\bar{i}+1}, \dots, A_{\bar{j}-1}$ are easily obtained by using the Cartesian tree. The minimum among them corresponds to the range minima problem restricted to the array of minima of the data in the processors. Thus we need a data structure to answer these queries in constant time. As the array of minima contains only p values, this data structure can be obtained by Algorithm 2 (section 3.2).

The difficulty of case 2.b is that we cannot construct the PS-tree explicitly in each processor as described in section 3.2, since this would require a memory of size $O(p \log p)$, larger than the memory requirement in the CGM model, which is $O(\frac{n}{p})$, with $\frac{n}{p} \geq p$. To overcome this difficulty we construct arrays P' and S' of $\log p+1$ positions each, described as follows, to store some partial information of the PS-tree T in each processor. Let us describe this construction in processor i, with $0 \leq i \leq p-1$. Let b_i be the value of the minimum of array A_i. Let v be any vertex of T such that the subtree with root v has b_i as leaf and let d_v be the **depth of v** in T, that is the path length from the root to v, as defined in [1]; and l_v be the **level of v**, which is the height of the tree minus the depth of v, as defined in [1] ($l_v = \log p - d_v$, because tree T has height $\log p$). Array P' (respectively, S') contains in position l_v the value of array P_v (respectively, S_v), of the level l_v of T, in the position corresponding to the leaf b_i. In other words, we have $P'[l_v] = P_v[i \bmod 2^{l_v} + 1]$.

Figure 3 illustrates the correspondence between arrays P' and S' stored in each processor and the PS-tree constructed by the algorithm of Alon and Schieber [2]. In Figure 3.b, the bold face positions in arrays S in each level of the tree correspond to the suffix minimum of $b_0 = 3$ in each level. In this way, we obtain array S' in P_0 (figure 3.c).

ALGORITHM 3: Range Minima (CGM model)
{Each processor receives $\frac{n}{p}$ contiguous positions of array A, partitioned into subarrays $A_0, A_1, \ldots, A_{p-1}$.}

1. Each processor i executes sequentially Algorithm 1 (section 3.1).
2. {Each processor constructs an array $B = (b_i)$ of size p, that contains the minimum of the data stored in each processor.}
   - 2.1. Each processor i calculates $b_i = \min A_i = \min\{a_{i\frac{n}{p}+1}, \ldots, a_{(i+1)\frac{n}{p}}\}$.
   - 2.2. Each processor i sends b_i to the other processors.
   - 2.3. Each processor i puts in b_k the value received from processor k, $k \in \{0, \ldots, p-1\} \setminus \{i\}$.
3. Each processor i executes procedures Construct_P' and Construct_S' (see below) .

   {By observing the description of Algorithm 2 (section 3.2), $P'[k]$ contains position i of the prefix minimum array in level k, $0 \leq k \leq \log p$.}

Given array B, the following procedure constructs array P' of $\log p + 1$ postitions in processor i, for $0 \leq i \leq p-1$. This procedure constructs P' in $O(p)$ ($= O(\frac{n}{p})$) time using only local data. The construction of array S' is done in a symmetric way, considering array B in reverse order.

PROCEDURE 4: Construct_P'.

1. $P'[0] \leftarrow b_i$
2. $pointer \leftarrow i$
3. $inorder \leftarrow 2 * i + 1$
4. **for** $k \leftarrow 1$ **until** $\log p$ **do**
5. $\quad P'[k] \leftarrow P'[k-1]$

6. **if** $\lfloor inorder/2^k \rfloor$ is odd
7. **then for** $l \leftarrow 1$ **until** 2^{k-1} **do**
8. $pointer \leftarrow pointer - 1$
9. **if** $P'[k] > B[pointer]$
10. **then** $P'[k] \leftarrow B[pointer]$

To simplify the correctness proof of this procedure, we consider p to be a power of 2 and that, in each processor i, array B contains leaves of a complete binary tree, as in the description of the sequential algorithm of section 3.2. We do not store an entire array in each internal node of this tree, but only the $\log p + 1$ positions of the prefix minimum array in each level corresponding to position i of array B.

The value of the variable *inorder* of line 3 is the value of the in-order number of the leaf containing b_i, obtained in an in-order traversal of the tree nodes. It can easily be seen that these values, from the left to the right, are the odd numbers in the interval $[1,\, 2p-1]$.

Theorem 1 *Procedure 4 correctly calculates array P', for each processor i, $0 \leq i \leq p-1$.*

Idea of proof. For a processor i, $0 \leq i \leq p-1$, we prove the following invariant at each iteration of the for loop of line 4: At each iteration k, let T_k be subtree that contains b_i and has root at level k. $P'[k]$ stores the position i of the minimum prefix array of the subarray of B corresponding to the leaves of subtree T_k and the variable *pointer* contains the index, in B, of the leftmost leaf of T_k. □

To determine S' we have a similar theorem with a similar proof.

Lemma 1 *Execution of Procedure 4 in each processor takes $O(\frac{n}{p})$ sequential time.*

Proof. The maximum number of iterations is $\log p$. In each iteration, the value of *pointer* is decremented or remains the same. The worst case is when $i = p-1$. In this case, at each iteration the value of the variable *pointer* is updated. As the number of elements of B is p, the algorithm updates the value of *pointer* at most p times. Therefore, procedure 4 is executed in $O(p) = O(\frac{n}{p})$ sequential time. □

Theorem 2 *Algorithm 3 solves the range minima problem in $O(\frac{n}{p})$ time using $O(1)$ communication rounds.*

Proof. Step 1 is executed in $O(\frac{n}{p})$ sequential time and does not use communication. Step 2 runs in $O(\frac{n}{p})$ sequential time and uses one communication round. By theorem 1, step 3 is executed in $O(\frac{n}{p})$ sequential time and does not require communication. Therefore, algorithm 3 solves the range minima problem in $O(\frac{n}{p})$ time using $O(1)$ communication rounds. □

5 Query processing

In this section, we show how to use the output of algorithm 3 to answer queries $MIN(i,j)$ in constant time. A query $MIN(i,j)$ is handled as follows. Assume that i and j are known by all the processors and the result will be given by processor 0. If a_i and a_j are in a same processor, then $MIN(i,j)$ can be determined by step 1 of algorithm 3. Otherwise, suppose that a_i and a_j are in distinct processors $p_{\bar{i}}$ and $p_{\bar{j}}$, respectively, with $\bar{i} < \bar{j}$. Let $right(\bar{i})$ the index in A of the rightmost element in array $A_{\bar{i}}$, and $left(\bar{j})$ the index in A of the leftmost element in array $A_{\bar{j}}$. Calculate $MIN(i, right(\bar{i}))$ and $MIN(left(\bar{j}), j)$, using step 1. We have two cases:

1. If $\bar{j} = \bar{i} + 1$ then $MIN(i,j) = \min\{MIN(i, right(\bar{i})), MIN(left(\bar{j}), j)\}$.

2. If $\bar{i} + 1 < \bar{j}$, then calculate $MIN(right(\bar{i}) + 1, left(\bar{j}) - 1)$, using step 3 of the algorithm. Notice that $MIN(right(\bar{i}) + 1, left(\bar{j}) - 1)$ corresponds to $\min\{b_{\bar{i}+1}, \ldots, b_{\bar{j}-1}\}$. Thus, $MIN(i,j) = \min\{MIN(i, right(\bar{i})), MIN(right(\bar{i}) + 1, left(\bar{j}) - 1), MIN(left(\bar{j}), j)\}$.

The value of $MIN(right(\bar{i}) + 1, left(\bar{j}) - 1)$ is obtained using step 3, as follows. Each processor calculates $w = \mathrm{LCA}(b_{\bar{i}+1}, b_{\bar{j}-1})$ in constant time. Then determines the level l_w; determines v and u, the left and right sons of w, respectively; determines l, the index of the leftmost leaf in the subtree of u. Processor $\bar{i} + 1$ calculates $S_v(2^{l_w - 1} - l + \bar{i})$ and sends this value to processor 0. Processor $\bar{j} - 1$ calculates $P_u(\bar{j} - l)$ and sends this value to processor 0. Processor 0, finally, calculates the minimum of the received minima. In both cases, processor 0 receives the minima of processors $\bar{i}$ and $\bar{j}$.

Notice finally that the correctness of Algorithm 3 results from the observations presented in this section.

6 Concluding remarks

In the CGM(n,p) model, we are interested in designing parallel algorithms that have linear local executing time and that utilize few communication rounds – preferably a constant number. This is because the communication between processors is more expensive than the computing time of the local data.

In this work we present an algorithm, in the CGM(n,p) model, for the range minima problem that runs in $O(\frac{n}{p})$ time using $O(1)$ communication rounds. This algorithm can be used to design a CGM algorithm for the LCA problem, based on the PRAM algorithm of JáJá [12], instead of other PRAM algorithms as in [13, 16] which would give less efficient implementations.

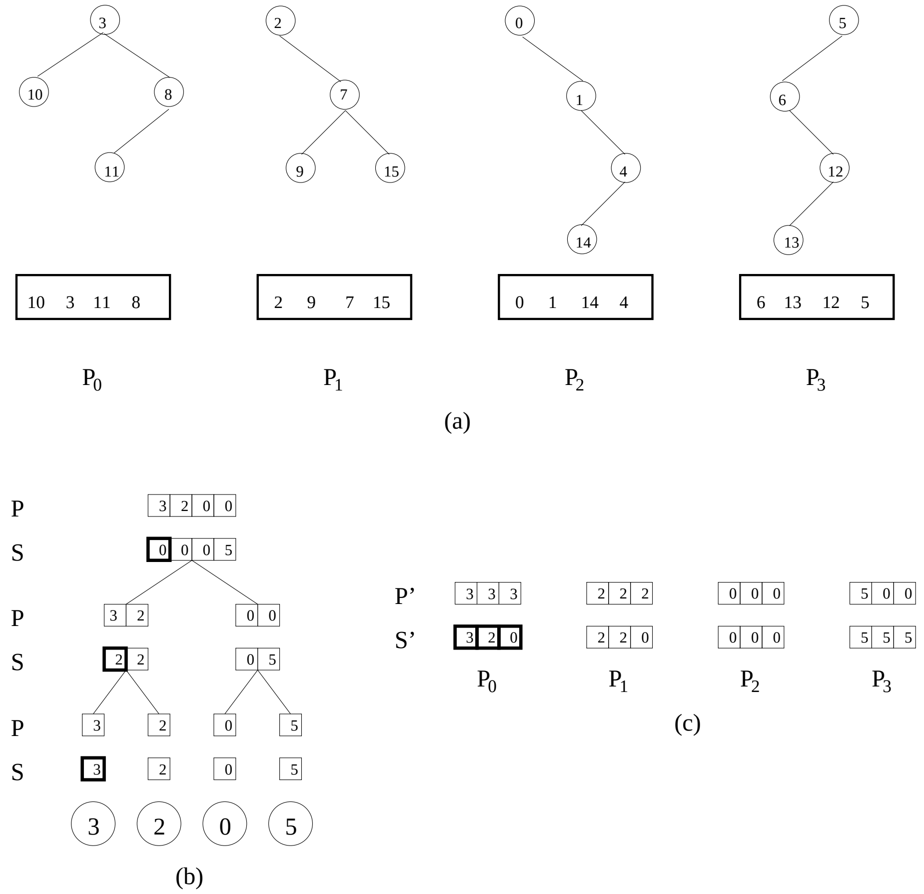

Fig. 3. Execution of algorithm 3 using the array $(10, 3, 11, 8, 2, 9, 7, 15, 0, 1, 14, 4, 6, 13, 12, 5)$. (a) The data distributed in the processors and the corresponding Cartesian trees. (b) PS-trees constructed by the algorithm of Alon and Schieber [2] of section 3.2 for the array $(3, 2, 0, 5)$ of the minima of processors. (c) Arrays P' and S' constructed by step 3 of algorithm 3 corresponding to arrays P and S of T.

References

1. A. V. Aho, J. E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1975.
2. N. Alon and B. Schieber. Optimal preprocessing for answering on-line product queries. Technical Report TR 71/87, The Moise and Frida Eskenasy Inst. of Computer Science, Tel Aviv University, 1987.
3. A. Amir, G. M. Landau, and U. Vishkin. Efficient pattern matching with scaling. *Journal of Algorithms*, 13:2–32, 1992.
4. O. Berkman, B. Schieber, and U. Vishkin. Optimal doubly logarithmic parallel algorithms based on finding all nearest smaller values. *Journal of Algorithms*, 14:344–370, 1993.
5. O. Berkman and U. Vishkin. Recursive star-tree parallel data structure. *SIAM J. Comput.*, 22(2):221–242, 1993.
6. E. Cáceres, F. Dehne, A. Ferreira, P. Flocchini, I. Rieping, A. Roncato, N. Santoro, and S. W. Song. Efficient parallel graph algorithms for coarse grained multicomputers and BSP. In *Proc. ICALP'97*, 1997.
7. F. Dehne, A. Fabri, and C. Kenyon. Scalable and architecture independent parallel geometric algorithms with high probability optimal time. In *Proc. 6th IEEE Symposium on Parallel and Distributed Processing*, pages 586–593, 1994.
8. F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable parallel computational geometry for coarse grained multicomputers. In *Proc. ACM 9th Annual Computational Geometry*, pages 298–307, 1993.
9. F. Dehne and S. W. Song. Randomized parallel list ranking for distributed memory multiprocessors. In *Proc. Second Asian Computing Science Conference (ASIAN'96)*, pages 1–10, 1996.
10. H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proc. 16th ACM Symp. On Theory of Computing*, pages 135–143, 1984.
11. D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–335, 1984.
12. J. Jájá. *An Introduction to Parallel Algorithms*. Addison-Wesley Publishing Company, 1992.
13. R. Lin and S. Olariu. A simple optimal parallel algorithm to solve the lowest common ancestor problem. In *Proc. of International Conference on Computing and Information - ICCI'91*, number 497 in Lecture Notes in Computer Science, pages 455–461, 1991.
14. V. L. Ramachandran and U. Vishkin. Efficient parallel triconectivity in logarithmic parallel time. In *Proc. of AWOC'88*, number 319 in Lecture Notes in Computer Science, pages 33–42, 1988.
15. J. H. Reif, editor. *Synthesis of Parallel Algorithms*. Morgan Kaufmann Publishers, 1993.
16. B. Schieber and U. Vishkin. On finding lowest common ancestors: simplification and parallelization. *SIAM J. Comput.*, 17:1253–1262, 1988.
17. L. G. Valiant. A bridging model for parallel computation. *Comm. of the ACM*, 33:103–111, 1990.
18. J. Vuillemin. A unified look at data structures. *Comm. of the ACM*, 23:229–239, 1980.