AMO - Advanced Modeling and Optimization, Volume 9*, Number* 1*,* 2007

Parallel Generation of the Biological Trees [1]

H. Ahrabian[a,b,c], A. Nowzari-Dalini[a,b,c], M. Razaghi[c], F. Zare-Mirakabad[c]
[a] *Center of Excellence in Biomathematics,*
School of Mathematics, Statistics, and Computer Science,
University of Tehran, Tehran, Iran.
[b] *Institute for Studies in Theoretical Physics and Mathematics (I.P.M.),*
Tehran, Iran.
[c] *Department of Bioinformatics,*
Institute of Biochemistry and Biophysics,
University of Tehran, Tehran, Iran.
E-mail: {ahrabian,nowzari}@ut.ac.ir and {razaghi,zare}@ibb.ut.ac.ir

Abstract

Several biological features are presented by different types of trees. Two types of such trees are considered in this paper. the first type is trees with n external nodes that each internal node have at least two children, and are used in neuro-science and called neuronal dendritic trees. The second type is trees with n internal nodes and m external nodes. This type of trees represent the secondary structure of RNA sequences, and called RNA trees. In this paper, we present two new parallel algorithms for generation of these two biological trees. Both algorithms are adoptive and cost-optimal and generate the trees in B-order. Computations run in an SM SIMD model.

Keywords*:* Neuronal Dendritic Trees, RNA Trees, Parallel Algorithm, B-order.

1 Introduction

One of the most basic and simple data structures in computer science, that are used in many other sciences such as biological science, is trees. Trees can be used to maintain any ordered set that must be accessed and updated. For many reasons, it is often useful to have available lists of all the shapes of trees of a certain type. Thus, many papers have appeared which contain sequential algorithms for generating trees [8, 13, 17, 18, 20, 26]. In most of these algorithms, trees are encoded as integer sequences and then these sequences are generated with a certain order and consequently their corresponding trees are also generated in a specific order. The most well-known orderings on trees are A-order and

[1] This research was in part supported by a grant from I.P.M. (No. 84920018).

B-order [27], and some of well-known encodings are P-sequences [14], inversion table [10], 0-1 sequences [27].

An optimal sequential generation algorithm takes constant average time to generate all trees. Using parallel techniques we can improve the speed between the generation of any two consecutive trees [3]. Recently, in this order, few parallel algorithms for generation of trees have been published for various parallel models [1, 2, 4, 5, 12, 21, 22, 23, 24].

Here, we consider two biological trees, neuronal dendritic trees and RNA trees. A simple, efficient sequential algorithm is presented by Pallo [15] for generating the codewords of all neuronal dendritic trees with a given number of external nodes. The corresponding trees are generated in B-order. Also Pallo [14] introduced a coding and a sequential generation algorithm for producing all RNA trees with n internal nodes and m external nodes. Similarly, the corresponding trees are generated in B-order. Up to now, no parallel algorithm is given in the literature for these trees.

In this paper, we present two cost-optimal and adaptive parallel algorithms for generation of dendritic trees and RNA trees. The algorithms are the parallel version of corresponding sequential algorithms given in [14, 15]. The computational model for both algorithms are SM SIMD computer [3].

The paper is organized as follows. Section 2 introduces the definitions and notions that is used further. In Section 3, we introduce a new parallel generation algorithm for neuronal dendritic trees. The parallel algorithm for RNA trees is given in Section 4. Finally, some concluding remarks are offered in Section 5.

2 Definitions

In a rooted, ordered tree every node except the one has a parent. The node without parent is called the *root* of tree. Every node has $r \geq 0$ children (the order is significant) and each of these children is also a tree called subtree of this node. The number of subtrees of a node is called the degree of that node. A node of degree zero is called an external node of leaf. A non-external node is called an internal node. A t-ary tree with n internal nodes is a ordered tree, in which every internal node has exactly t ordered children. Clearly, an n-node t-ary tree has $(t-1)n+1$ external nodes. The set t-ary trees with n internal nodes is denoted by $T_{n,t}$ [11].

As it is mentioned, in tree generation terminology, trees are encoded as integer sequences and then these sequences are generated with certain order and consequently their corresponding trees are generated in a specific order. Such an ordering is B-order which is defined for an arbitrary trees as follows [27].

Definition 1 *Given a tree T, let r_T be the degree of its root and T_i be the subtree rooted at the ith son of the root of T. We say that T and T' are in B-order ($T \prec_B T'$) if*

1. *$r_T < r_{T'}$, or*

2. *$r_T = r_{T'}$, and for some i $(1 \leq i \leq r_T)$ we have*

(a) $T_j = T'_j$ *for* $j = 1, 2, \cdots, i-1$, *and*

(b) $T_i \prec_B T'_i$.

The most well-defined ordering for integer sequences is lexicographic ordering which is defined as follows.

Definition 2 *The two integer sequence* $v = (v_1, v_2, \cdots, v_n)$ *and* $v' = (v'_1, v'_2, \cdots, v'_m)$ *are in lexicographic order, and will denote* $v < v'$, *if there exist* $i \in [1, min(n, m)]$ *such that*

1. $v_j = v'_j$ *for all* $j \in [1, i-1]$,
2. $v_i < v'_i$.

As it is mentioned, we consider two biological trees, neuronal dendritic trees and RNA trees. The term *neuronal tree* is used to refer a tree for which a variable degree of each internal node is greater or equal to 2 [6, 15]. Usually, these trees are characterized with fixed amount of external nodes. For example a tree given in Figure 1 can be regarded as a neuronal tree with $n = 13$ external nodes. Translated into dendritic terminology, the root is taken to be the axon hillock and the external nodes are the tips of the terminal segments. The order of magnitude of branching at a node may be described as dichotomous if the degree of that node is 2, trichotomous if the degree is 3, and so on [7, 15].

On the other hand, RNA tree refer to a tree which has n internal nodes, and m external nodes. For example a tree given in Figure 1 can be regarded as a RNA tree with $n = 5$ and $m = 13$. These trees represent the secondary structure of a RNA sequence of length $2n - 2 + m$ with $n - 2$ base pairs. RNA is a chain molecule, mathematically a string over a four letter alphabet. It is built from a nucleotides containing bases A (adenine), C (cytosine), G (guanine) and U (uracil). These bases can form base pairs, conventionally A pairs with U and C pairs with G. These are called Watson-Crick pairs. By folding

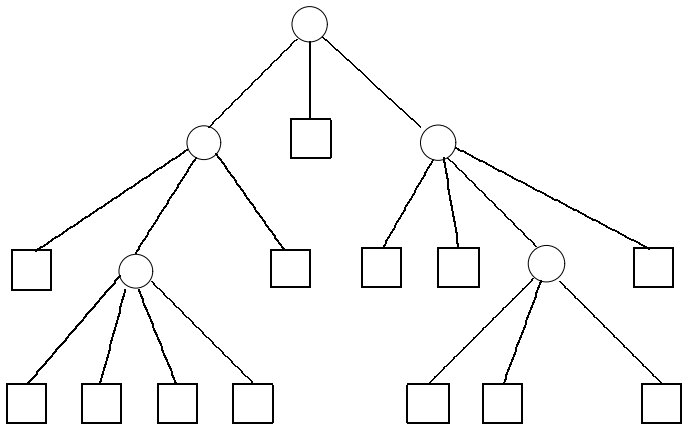

Figure 1: A sample tree with 5 internal nodes and 13 external nodes.

back into itself, an RNA molecule forms a structure, and is stabilized by the forms of hydrogen bonds between certain pairs of bases and dense stacking of neighboring base pairs. See Figure 2 (a) for an example of a so-called cloverleaf structure of sample RNA. In Figure 2 (b), the alternative representation of these RNA is shown. The primary structure is written along the horizontal axis and the base pairs are shown as arcs [25].

Now, we illustrate the bijection relation between the tree and the secondary structure of RNA sequences with an example given in Figure 3 (a). We put a node above the outside of all loops as the root of tree. Then inside of each arcs we insert a node as internal nodes of tree. Each internal node is connected to the node in the upper loop as its parent, and to any unpaired base in its corresponding loops as its children. These unpaired bases are considered as external nodes of tree. With regard to this bijection, the tree corresponding to secondary structure of the RNA sequence given in Figure 3 (a) is shown in Figure 3 (b).

3 Neuronal dendritic trees

In this section, we review the concepts introduced in [15] for neuronal trees such as sequential generation algorithm, that we need further for designing our parallel algorithm for generation of neuronal trees given later.

Let S_n denotes the set of neuronal trees with n external nodes. The number of trees in S_n (*i.e.*, $|S_n|$) is the well-known nth Schröder number [19] and can be computed by a

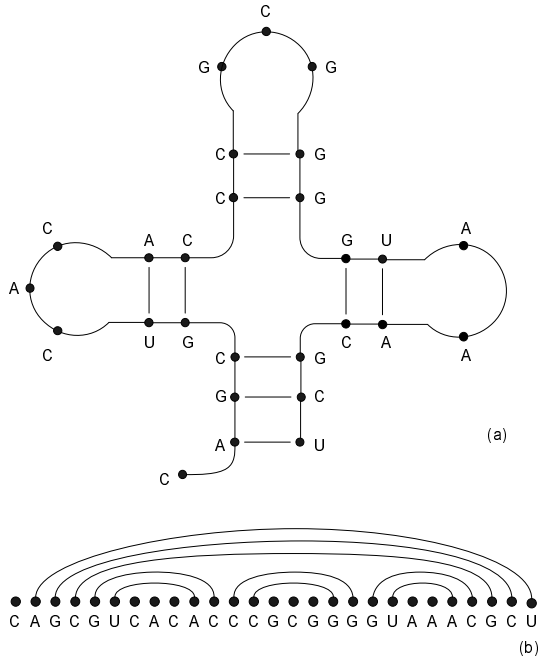

Figure 2: Two different representation for the secondary structure of a sample RNA.

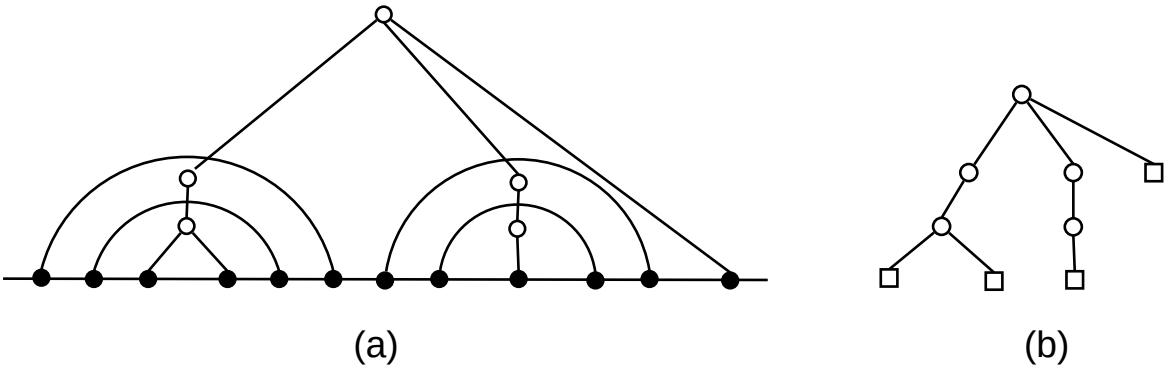

Figure 3: Trees corresponding to the secondary structure of the sample RNA .

linear recurrence formula.

Theorem 1 *The Schröder number counts the trees of S_n as follows:*

$$|S_n| = \frac{3(2n-3)|S_{n-1}| - (n-3)|S_{n-2}|}{n}, \quad \text{for} \quad n > 2,$$

$$|S_1| = |S_2| = 1.$$

Each tree in the set S_n can be encoded as an integer sequences based on the following definition [15].

Definition 3 *Given a neuronal tree T with n external nodes, the S-sequences $s = \{s_1, s_2, \cdots, s_\ell\}$ is obtained by labeling each internal node of tree with its degree minus one and each external node with zero, and the labels are the listed in pre-order traversal of T as a sequence.*

For example the S-sequence corresponding to the tree T denoted in Figure 1 is the sequence $s = \{2, 2, 0, 3, 0, 0, 0, 0, 0, 0, 3, 0, 0, 2, 0, 0, 0, 0\}$.

The properties of a sequence corresponding to a neuronal tree is given in the following theorem [15].

Theorem 2 *An integer sequence $s = \{s_1, s_2, \cdots, s_\ell\}$ is a feasible codeword of a neuronal tree iff $s_\ell = 0$, and for $k \in [1, \ell - 1]$*

$$\sum_{i=1}^{k} s_i > |\{j \in [1, k] : s_j = 0\}|.$$

Clearly, there is one to one correspondence between a neuronal tree and feasible codewords. With respect to the above theorem, the length of feasible sequences alters between

$2n-1$ to $n+1$. In the corresponding lexicographic ordering of the S-sequence, the first S-sequence is

$$\{\underbrace{1,0,1,0,\cdots,1,0}_{2n-2},0\},$$

with length $2n-1$, and the last is

$$\{n-1,\underbrace{0,0,\cdots,0}_{n}\},$$

with length $n+1$. Actually, the first sequence corresponds to a right-chain binary tree with $n-1$ internal nodes and n external nodes, and the last sequence corresponds to a n-ary tree with one internal node and n external nodes. Therefore, recall from [15], we can write the following theorem.

Theorem 3 *Given two neuronal trees T and T' belong to the set S_n, T and T' are in B-order, $T \prec_{_B} T'$, iff s_T is lexicographically less than $s_{T'}$.*

As it is mentioned, the sequential generation algorithm for S-sequences in B-order is given by Pallo [15]. Pallo postulated an algorithm that returns the successor of a given sequence $s = \{s_1, s_2, \cdots, s_\ell\}$ with length $\ell = O(n)$. In this algorithm, the sequence s is a scanned from right to left and the first non-zero element is obtained, and its position is assigned in k. If $k = 1$ then this sequence corresponds to the last sequence and there is no successor. Otherwise, the successor is computed as follows. First the length of the new sequence is evaluated and this length is kept in ℓ. Later, the $k-1$th element is incremented and a subsequence corresponding to a right-chain subtree with size $s_k - 1$ is replaced by the last $s_k - 1$ elements in the sequence and the elements from k to $\ell - 2s_k + 1$ are set to zero. In fact, this process is similar to the replacement of the right-most child of the node k by a right-chain binary subtree, and the appropriate number of external nodes is added as the first children of the kth node. Clearly, the running time of this algorithm is $O(n)$ in worst-case.

Now, we present a parallel version of the above sequential algorithm. This algorithm is illustrated in Figure 4 and similar to the sequential version, this algorithm generates the successor sequence of a given tree sequence $s = \{s_1, s_2, \cdots, s_\ell\}$. Our computational model is a CREW SM SIMD with $N \leq \ell$ processors (ℓ is a length of longest S-sequence), and it can be proved that the algorithm is cost-optimal and adaptive.

Initially, the value of ℓ must be made known to all N processors. This can be done by using the procedure broadcast in $O(\log N)$ time complexity. Later, the S-sequence is subdivided to N subsequences of length $d = \lceil \ell/N \rceil$ and each processor i is assigned to the subsequence $\{s_{(i-1)\times d+1}, s_{(i-1)\times d+2}, \cdots, s_{i\times d}\}$, where $1 \leq i \leq N$. Each processor i finds the position of right-most non-zero value and stores it in variable g_i. This process requires $O(\ell/N)$ time complexity. Then between all the N computed values g_i, the maximum position is evaluated. This process can be done by algorithm parallel maximum [3] in $O(\log N)$. Let k be this maximum position with value u. Now, the length of the new sequence is computed by one of processors (*e.g.*, processor 1), $(k-1)$th element is incremented, and the

```
Procedure Parallel-Next-Sseq (s : Sseq) ;
Var  i, j, k, u, q, d : Integer ; g : Array [1 .. N] of Integer ;
Begin
    d := ⌈ℓ/N⌉ ;
    For i := 1 To  N Do In Parallel
        g_i := 0 ;
        For j := i × d DownTo  (i − 1) × d + 1 Do
            If (j ≤ ℓ) And (s_j ≠ 0) And (g_i = 0) Then
                g_i := j ;
    End ;
    ParallelMax (g, k) ;   u := s_k ;
    If (k = 1) Then Exit ;
    If (s_{k−1} = 0) Then   ℓ := ℓ + u − 1 ;
    Else    ℓ := ℓ + u − 2 ;
    s_{k−1} := s_{k−1} + 1 ;   s_l := 0 ;
    BroadCast (k, u, ℓ) ;
    d := ⌈u − 1/N⌉ ;
    For i := 1 To  N Do In Parallel
        For j := i × d DownTo  (i − 1) × d + 1 Do
            If (j ≤ u − 1) Then Begin
                s_{l−2j} := 1 ;   s_{l−2j+1} := 0 ;
            End ;
    End ;
    d := ⌈(l − 2u − k + 2)/N⌉ ;
    For i := 1 To  N Do In Parallel
        For j := i × d DownTo  (i − 1) × d + 1 Do
            If ((j + k − 1) ≤ (l − 2u + 1)) Then
                s_{j+k−1} := 0 ;
    End ;
End ;
```

Figure 4: Parallel algorithm *Parallel-Next-Sseq*.

three values k, u, ℓ are broadcasted to all processors in $O(\log N)$ time. Later, the sequence $\{s_{\ell-2u+2}, \cdots, s_\ell\}$ are subdivided into N subsequences of length $d = \lceil \ell - 2u + 2/N \rceil$, and each processor i is assigned to the subsequence $\{s_{(i-1)\times d+1}, s_{(i-1)\times d+2}, \cdots, s_{i\times d}\}$, where $1 \leq i \leq N$. All the processors in parallel replace the sequence $\{s_{\ell-2u+2}, \cdots, s_\ell\}$ with a sequence corresponding to the right-chain binary tree of size $u - 1$. This operation is performed in $O(\ell/N)$. Again the sequence $\{s_k, \cdots, s_{\ell-2u+1}\}$ is set to zero in parallel in $O(\ell/N)$, and the successor sequence is obtained.

With regard to the time complexity of the above steps, the total required time for this algorithm is $T(n) = O(\ell/N + \log\ N)$, and because $l = 2n - 1$ in worst-case therefore

$T(n) = O(n/N + \log\ N)$. Now we can easily prove that the algorithm is cost-optimal and adaptive.

Theorem 4 *The presented Parallel-Next-Sseq is cost-optimal and adaptive.*

Proof. Considering the time complexity of *Parallel-Next-Sseq*, the cost of this algorithm is equal to $C(n) = O(n + N\ \log\ N)$. Thus, with regard to the time complexity of the sequential algorithm which is $O(n)$, the parallel algorithm is cost-optimal for $N \leq n/log\ n$. The adaptivity of this algorithm is clear. ■

4 RNA trees

In this section, we present a parallel generation algorithm for RNA trees. For this reason, we first review few concepts introduced in [14, 25] that we need further. Later, Pallo's sequential generation algorithm [14] that our parallel algorithm is based on, is discussed.

Let $L_{n,m}$ be the set of all order trees with n internal nodes and m external nodes. The number of these trees can be counted by Narayana numbers [16].

Theorem 5 *The Narayana numbers counts the trees of $L_{n,m}$ as follows:*

$$|L_{n,m}| = \frac{1}{n+m-1}\binom{n+m-1}{m}\binom{n+m-1}{m-1}$$

.

Now the encoding given by Pallo [14] for trees with n internal nodes and m external nodes is described. For this purpose, the P-sequences introduced for t-ary trees in [14], are used for m-ary trees, such that the trees with n internal nodes and m external nodes are embedded in the set of regular, m-ary trees with n internal nodes.

Let us define $\overline{T}_{n,m}$ as the set of all regular m-ary trees with n internal nodes and with two kinds of external nodes: m *real* external nodes □, and $(n-1)(m-1)$ *virtual* external nodes ■. This definition is justified by the following injective embedding, $\varnothing : L_{n,m} \rightarrow \overline{T}_{n,m}$, such that for each internal node with degree $r < m$, $m - r$ *virtual* external nodes are attached as a $m - r$ first children. In Figure 5, a 4-ary tree corresponding to the tree given in Figure 3 (b) is shown.

Now, we introduce P-sequence for regular t-ary tree, and then we extend this definition for $\overline{T}_{n,m}$, as $\overline{\text{P}}$-sequence [14].

Definition 4 *The P-sequence of a t-ary tree T with n internal nodes is the integer sequence $\{p_1, p_2 \cdots p_{n(t-1)}\}$, where p_i is the number of internal nodes written before external node i in pre-order traversal of T. Since the integer corresponding to the last external node, the $n(t-1)+1$th external node, is always equal to n therefore we omit it.*

Let us denote $\overline{\mathbb{N}} = \{\overline{n}, n \in \mathbb{N}\}$ and $|\overline{n}| = |n| = n$, then with regard to the definition of the P-sequences, the $\overline{\text{P}}$-sequences for encoding the trees in the set $\overline{T}_{n,m}$ are defined as follows [14].

Definition 5 *Given a tree $T \in \overline{T}_{n,m}$, the $\overline{P}$-sequence of T is the sequence $\{p_1, p_2 \cdots p_{n(m-1)}\}$ defined by: If n_i is the number of internal nodes written before the external node i in pre-order traversal of T, then $p_i = n_i$ if the external node i is real external node, and $p_i = \overline{n_i}$ if the external node i is virtual external node.*

For example the $\overline{\mathrm{P}}$-sequence corresponding to the tree T denoted in Figure 5 is the sequence $p = \{\overline{1}, \overline{2}, \overline{2}, \overline{2}, \overline{3}, \overline{3}, 3, 3, \overline{4}, \overline{4}, \overline{4}, \overline{4}, \overline{5}, \overline{5}, \overline{5}, 5\}$. Actually, the integers with symbol $'-'$ are the labels of the *virtual* external nodes, and the integers without symbol $'-'$ are the labels of the *real* external nodes.

Now, the properties of a feasible sequence encoding a tree $T \in L_{n,m}$ is given [14].

Theorem 6 *The sequence $\{p_1, p_2 \cdots p_{n(m-1)}\}$ on $\mathbb{N} \cup \overline{\mathbb{N}}$ is the $\overline{P}$-sequence of a tree $T \in L_{n,m}$ iff:*

1. *The sequence obtained by deleting the symbol $'-'$ is the P-sequence of a tree $T' \in T_{n,m}$.*
2. *$|\{i \in [1, n(m-1)] : p_i \in \overline{\mathbb{N}}\}| = (n-1)(m-1)$.*
3. *For all $\ell \in [1, n]$, $|\{i \in [1, n(m-1)] : p_i = \overline{\ell}\}| \leq m-1$.*
4. *For all $k \in [1, n]$, if there exist i and j such that $p_i = k$ and $p_j = k$, then $j < i$.*

Let us define on $\mathbb{N} \cup \overline{\mathbb{N}}$ the following ordering:

$$\overline{1} < 1 < \overline{2} < 2 < \overline{3} < 3 \cdots < n < \overline{n+1} < n+1 < \cdots .$$

In the corresponding lexicographic ordering of the $\overline{\mathrm{P}}$-sequence, the first $\overline{\mathrm{P}}$-sequence is

$$\{\underbrace{\overline{1}, \cdots, \overline{1}}_{m-1}, \underbrace{\overline{2}, \cdots, \overline{2}}_{m-1}, \underbrace{\overline{n-1}, \cdots, \overline{n-1}}_{m-1}, \underbrace{n, \cdots, n}_{m-1}\},$$

and the last is

$$\{\underbrace{\overline{2}, \cdots, \overline{2}}_{m-1}, \underbrace{\overline{3}, \cdots, \overline{3}}_{m-1}, \underbrace{\overline{n}, \cdots, \overline{n}}_{m-1}, \underbrace{n, \cdots, n}_{m-1}\}.$$

Therefore, recall from [14], we can write the following theorem.

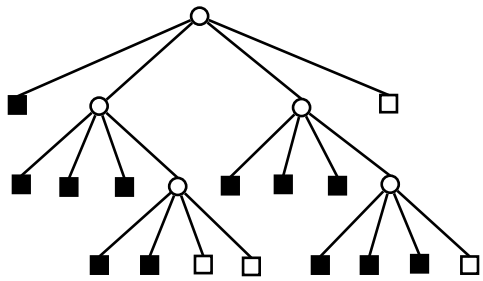

Figure 5: A sample embedded RNA tree.

Theorem 7 *Given two RNA trees T and T' belong to the set $L_{n,m}$, T and T' are in B-order, $T \prec_B T'$, iff p_T is lexicographically less than $p_{T'}$.*

As it is mentioned, the sequential generation algorithm for $\overline{\mathrm{P}}$-sequence in B-order is given by Pallo [14]. Pallo designed an algorithm that returns the successor of a given sequence $\{p_1, p_2, \cdots, p_\ell\}$ with length $\ell = n(m-1)$. In this algorithm, the codeword is scanned from right to left and the last *incrementable element* is searched. With regard to this point that it is assumed, $\overline{1} < 1 < \overline{2} < 2 < \overline{3} < 3 \cdots < \overline{n} < n$, therefore the elements are selected for increasing with this order. The *incrementable element*, is the element with the least index k such that $\{p_{k+1}, p_2, \cdots, p_\ell\}$ is a subsequent of the last $\overline{\mathrm{P}}$-sequence. Now, the number of elements p_j $(k < j \leq \ell)$ that belongs to $\overline{\mathbb{N}}$, is obtained and saved in the variable u. If the kth element belongs to $\mathbb{N}$, it is added by one and is changed to an element of $\overline{\mathbb{N}}$, and u is decremented, otherwise the kth element loses its symbol $'-'$ and is changed to an element in $\mathbb{N}$. Then, each p_j $(k < j \leq \ell)$ is set to $Max\ (|p_k|, 1 + \lfloor (j-1)/(m-1) \rfloor)$, where $|p_k|$ is p_k without the symbol $'-'$. Clearly the constructed sequence $\{p_{k+1}, p_2, \cdots, p_\ell\}$ is a P-sequence. Finally, k elements of p_j $(k < j \leq \ell)$ is changed to an element of $\overline{\mathbb{N}}$, with regard to the Theorem 6. Clearly, the running time of this algorithm is $O(nm)$ in worst-case.

Now, we present a parallel version of the above sequential algorithm. This algorithm illustrated in Figure 6 and similar to the sequential version, this algorithm generates the successor sequence of a given tree sequence $p = \{p_1, p_2, \cdots, p_\ell\}$. Our computational model is a CRCW SM SIMD with $N \leq \ell$ processors, and it can be proved that the algorithm is cost-optimal and adaptive.

Initially, the values n, m, and $\ell = n(m-1)$ must be made known to all processors. This can be done using procedure broadcast in $O(\log N)$ time. The $\overline{\mathrm{P}}$-sequence is subdivided into N subsequences of length $d = \lceil \ell/N \rceil$, and each processor i is assigned $\{p_{(i-1)\times d+1}, p_{(i-1)\times d+2}, \cdots, p_{i\times d}\}$, where $1 \leq i \leq N$. All processor now perform the following steps. Each processor i finds the position of right-most *incrementable element* and stores it, and betweens all the N computed value, the minimum position is evaluated and stored in variable k. This step can be performed in $O(\ell/N)$ by the parallel pattern matching algorithm [9]. If we can not find any *incrementable element*, then this sequence is considered as the last sequence and no successor sequence is defined. Now again, the subsequence $\{p_k, \cdots, p_\ell\}$ is subdivided between N processors. Each processor computes the number of elements in each subsequences that belong to $\overline{\mathbb{N}}$, and assigns in the variable u. This process is performed by the procedure *ParallelCard*, which is shown in Figure 7, in $O(\ell/N + \log N)$. Later, in $O(1)$ the *incrementable element* p_k, is incremented based on the order $\overline{1} < 1 < \overline{2} < 2 < \overline{3} < 3 \cdots < \overline{n} < n$. Then u is decremented and the values k and u are broadcasted to all processors in $O(\log N)$. Now, the subsequence $\{p_{k+1}, \cdots, p_\ell\}$ is subdivided between N processors, and each processor i $(1 < i \leq N)$ construct a subsequence without regarding the symbol $'-'$. With respect to this fact that each internal node should have maximum $m-1$ *virtual* external nodes, therefore k elements of the subsequences $\{p_{k+1}, \cdots, p_\ell\}$ are converted to the element with symbol $'-'$. For this process, the number of $'-'$ symbols is computed in parallel by each processor i and assigned

```
Procedure Parallel-Next-P̄seq (p : P̄seq) ;
Var i, j, k, u, d : Integer ; q : Array [1 .. N] of Integer ;
Begin
    k := ParallelPattenMatching (p) ;
    If (k > 0) Then u :=ParallelCard(p, k, ℓ);
    Else Exit;
    If (p_k ∈ ℕ̄) Then Begin
        p_k := p̄_k + 1 ;   u := u − 1 ;   q_{k+1} := 1 ;
    End
    Else Begin
        p_k := |p_k| ;   q_{k+1} := m − 1 ;
    End ;
    BroadCast (k, u) ;
    d := ⌈(ℓ − k)/N⌉ ;
    For i := 1 To N Do In Parallel
        For j := i × d + k DownTo (i − 1) × d + k + 1 Do
            p_j :=Max (|p_k|, 1 + ⌊(j − 1)/(m − 1)⌋) ;
    End ;
    For i := 1 To N Do In Parallel
        For j := (i − 1) × d + k + 1 To i × d + k Do
            If (j ≤ ℓ) Then If (p_j ≠ |p_{j−1}|) Then q_{j+1} := 1
                                Else If (q_j < m − 1) Then q_{j+1} := q_j + 1 ;
                                    Else q_{j+1} := q_j ;
    End ;
    For i := 1 To N Do In Parallel
        For j := (i − 1) × d + k + 1 To i × d + k Do
            If (u > 0) And (j ≤ ℓ) Then
                If (p_j ≠ |p_{j−1}|) Then Begin
                    p_j := p̄_j ;   u := u − 1 ;
                End
                Else If q_j < m − 1 Then Begin
                        p_j := p̄_j ;   u := u − 1 ;
                    End ;
    End ;
End ;
```

Figure 6: Parallel algorithm *Parallel-Next-$\overline{P}$seq*.

in q_i in $O(\ell/N)$. Later each processor i, adjusts the numbers of $'-'$ symbols with regard to the value of q_i. It should be noted that, by adding an element with symbol $'-'$, each processor decrements the value u in parallel. This process needs a concurrent write operation. Therefore, this step is the only step we might need our model to be concurrent

```
Function ParallelCard (p : P̄seq; f, e : Integer) ;
Var i, j, d : Integer ; g : Array [1..N] of Integer
Begin
        d := ⌈(f − t + 1)/N⌉ ;
        For i := 1 To N Do In Parallel
              g_i := 0 ;
              For j := i × d + f − 1 DownTo (i − 1) × d + f Do
                    If (j ≤ e) And (p_j ∈ N̄) Then
                          g_i := g_i + 1 ;
        End ;
        For i := 1 To N/2 Do In Parallel
              For j := 0 To ⌈log N⌉ − 1 Do
                    If (2i + 2^j − 1 ≤ N) And ((i − 1) mod 2^j = 0) Then
                          g_{2i−1} := g_{2i−1} + g_{2i+2^j−1} ;
        End ;
        Return g_1 ;
  End ;
```

Figure 7: Algorithm *ParallelCard*.

write. In this case the summation of the values are considered. This recent operation require $O(\ell/N)$ time.

With regard to the time complexity of the above steps, the total required time for this algorithm is $T(\ell) = O(\ell/N + \log\ N)$. Now we can easily prove that the algorithm is cost-optimal and adaptive.

Theorem 8 *The presented Parallel-Next-*$\overline{P}$*seq is cost-optimal and adaptive.*

Proof. Considering the time complexity of *Parallel-Next-*$\overline{P}$*seq*, the cost of this algorithm is equal to $C(\ell) = O(\ell + N\ log\ N)$. Thus, with regard to the time complexity of the sequential algorithm which is $O(\ell)$, the parallel algorithm is cost-optimal for $N \leq \ell/log\ \ell$. It should be noted that $\ell = nm$. The adaptivity of this algorithm is clear. ■

5 Conclusion

We have given two simple parallel algorithms for generating the neuronal trees and RNA trees. These algorithm are the first parallel algorithms for generation of these type of trees given in the literature. The algorithms generate the neuronal trees and RNA trees in B-order on a SM SIMD model. Our model supports the cost-optimality of the algorithm with different number of processors. The algorithms are adaptive and the number of processors are variable.

References

[1] H. Ahrabian and A. Nowzari-Dalini, Parallel generation of binary trees in A-order, *Parallel Comput.* **31** (2005), 948–955.

[2] H. Ahrabian and A. Nowzari-Dalini, Adaptive generation of t-ary trees in parallel, *Adv. Modeling Optim.* **8** (2006), 19–28.

[3] S. G. Akl, *The Design and Analysis of Parallel Algorithms*, Prentice Hall, Englewood Cliffs, 1989.

[4] S. G. Akl and I. Stojmenovic, Generating binary trees in parallel, in: *Proc.* 30*th Annual Allerton Conference on Communication*, Monticello, Illinois, 1992, pp. 135–147.

[5] S. G. Akl and I. Stojmenovic, Generating t-ary trees in parallel, *Nordic J. Comput.* **3** (1996), 63–71.

[6] S. Berger and L. Tucker, Binary tree representation of three-dimentional, recosytruted neuronal trees: a simple, efficient algorithm, *Comput. Methods Programs Biomed.* **23** (1986), 231–235.

[7] M. Berry and P. Bradley, The application of network analysis to the study of branching patterns of large dendritic fields, *Brain Res.* **109** (1976), 111–132.

[8] M. C. Er, Efficient generation of k-ary trees in natural order, *Comput. J.* **35** (1992), 306-308.

[9] Z. Galil, A constant-time optimal parallel string-matching algorithm, *J. ACM* **42** (1995), 908–918.

[10] G. Knott. A numbering system for binary trees. *Comm. ACM* **20** (1977), 113–115.

[11] D. E. Knuth, *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*, 2nd Ed., Addison-Wesley, Reading, 1973.

[12] Z. Kokosinski, On parallel generation of t-ary trees in an associative model, *Lecture Notes in Computer Science* **2328** (2002), 228–235.

[13] J. F. Korsh, A-order generation of k-ary trees with $4k-4$ letter alphabet, *J. Inform. Optim. Sci.* **16** (1995), 557-567.

[14] J. Pallo, Generating trees with n nodes and m leaves, *Intern. J. Comput. Math.* **21** (1987), 133-144.

[15] J. Pallo, A simple algorithm for generating neuronal dendritic trees, *Comput. Methods Programs Biomed.* **33** (1990), 165–169.

[16] H. Prodinger, A coresspondence between orderd trees and nondecresing partitions, *Discrete Math.* **46** (1983), 205–206.

[17] D. Roelants Van Baronaigien and F. Ruskey, Generating t-ary trees in A-order, *Inform. Process. Lett.* **27** (1988), 205–213.

[18] F. Ruskey, Generating t-ary trees lexicographically, *SIAM J. Comput.* **7** (1978), 424–439.

[19] E. Schröder, Vier combinatorische problem, *Z. Math. Phys.* **15** (1870), 361–376.

[20] E. Trojanowski, Ranking and listing algorithm for k-ary trees, *SIAM J. Comput.* **7** (1978), 492–509.

[21] V. Vajnovszki and J. Pallo, Parallel algorithms for listing well-formed parentheses strings, *Parallel Process. Lett.* **8** (1998), 19–28.

[22] V. Vajnovszki and C. Phillips, Two optimal parallel algorithms for generating P-sequences, in: *Proc. 9th International Conference on Parallel and Distributed Computing Systems* (eds. K. Yetongnon and S. Hairi), International Society for Computers and their Applications, Raleigh, 1996, pp. 819–821.

[23] V. Vajnovszki and C. Phillips, *Generating k-ary trees in parallel*, in: *Proc. High Performance Computing* (ed. B. Werner), IEEE Computer Society Press, Las Vegas, 1997, pp. 117–121.

[24] V. Vajnovszki and C. Phillips, Systolic generation of k-ary trees, *Parallel Process. Lett.* **9** (1999), 93–101.

[25] M. S. Waterman, *Introduction to Computational Biology*, CRC Press, New York, 1995.

[26] Z. Yongjin and W. Jianfang, Generating k-ary trees in lexicographic order, *Sci. Sin.* **23** (1980), 1219-1225.

[27] S. Zaks, Lexicographic generation of ordered tree, *Theoret. Comput. Sci.* **10** (1980), 63–82.