Journal of Sciences, Islamic Republic of Iran 16(2): 169-173 (2005) http://jsciences.ut.ac.ir
University of Tehran, ISSN 1016-1104

Parallel Generation of *t*-ary Trees

H. Ahrabian[*] and A. Nowzari-Dalini

Department of Mathematics and Computer Science, Faculty of Sciences, University of Tehran, Tehran, Islamic Republic of Iran

Abstract

A parallel algorithm for generating t-ary tree sequences in reverse B-order is presented. The algorithm generates t-ary trees by 0-1 sequences, and each 0-1 sequences is generated in constant average time $O(1)$. The algorithm is executed on a CREW SM SIMD model, and is adaptive and cost-optimal. Prior to the discussion of the parallel algorithm a new sequential generation with $O(1)$ average time complexity, and ranking and unranking algorithms with $O(t\,n)$ time complexity is also given.

Keywords: *t*-ary Trees; Parallel algorithm; B-order; 0-1 Sequences; Recursion

1. Introduction

A few parallel algorithms for t-ary trees are presented by Stojmonovic and Akl [5], Vajonovszki and Phillips [16,17], and Kokosinski [10]. In [5], trees are represented by an inversion table and the processor model is a linear array multiprocessor. The generated integer sequences corresponding to the t-ary trees of n nodes in this algorithm are of length n and the parallel algorithm is executed with n processor. In [16], trees are represented by 0-1 sequences and the algorithm is run on a shared memory multiprocessor. Vajonovszki and Phillips [17] also presented a parallel generating algorithm for t-ary trees represented by generalized P-sequences on a linear array. The latter two algorithms generate sequences of length tn with tn processor. Finally, Kokosinski [10] generated t-ary trees of n nodes by 0-1 sequences in parallel with an associative model with n processor.

The design of most parallel algorithms is based on the sequential versions of them in the literature. There exist several sequential algorithms for generating t-ary trees [1,3,6,8,12-15,18,19].

In this paper we describe a parallel algorithm for generating t-ary tree sequences in reverse B-order. This algorithm is executed on a CREW SM SIMD model [4] and is adaptive and cost-optimal, and the number of processors can be much less than the other parallel generation algorithms. As it is mentioned, all the previous parallel algorithms for t-ary trees generate by 0-1 sequences are not presented as an adaptive algorithm and the number of processors employed in their models is of a fixed sized. Prior to the discussion of our adaptive parallel algorithm a new sequential generation, ranking and unranking algorithms are also given. This sequential algorithm generates each 0-1bit sequence in constant average $O(1)$ and the time complexity of ranking and unranking algorithm is $O(tn)$.

The paper is organized as follows. Section 2 introduces the definitions and defines the notions to use further. In Section 3, we introduce a new sequential generation algorithm with ranking and unranking algorithms. The parallel version of the sequential generation algorithm is given in Section 4. Finally, some concluding remarks are offered in Section 5.

[*] *E-mail: ahrabian@ut.ac.ir*

2. Definitions

The t-ary tree is a data structure consisting of a finite set of n nodes which either empty ($n=0$) or consists of a root and t disjoint children, and each child is a t-ary subtree, recursively defined. A node is the parent of another node if the latter is a child of the former. In a t-ary tree an external node is a node without child and an internal node is a node with exactly t children. An n-node t-ary tree has $(t-1)n + 1$ leaves, and the total number of t-ary trees with n internal nodes is denoted by $C_{n,t}$ and is known to have the value $C_{n,t} = \frac{1}{(t-1)n+1}\binom{tn}{n}$.

Let us introduce basic notations used through this paper and a definition of t-ary trees by means of choice functions of indexed families of sets [7,10,11].

Let $< A_i >_{i\in I}$ denote an *indexed family* of *sets* $A_i = A$, where $A= \{1, \ldots, m\}$, $I= \{1, \ldots, l\}$, and $l, m \geq 1$. Any mapping f which chooses one element from each set $A_1, \ldots, A_l$ is called a *choice function* of the family $<A_i>_{i\in I}$ [11]. With additional restrictions we can model by choice functions various classes of combinatorial objects [7,9]. If $A_i =\{0, 1\}$ and $I=\{1, \ldots, l\}$, then any choice function $\chi = < x_i >_{i\in I}$, that belong to the indexed family $< A_i >_{i\in I}$, is called *binary choice function* of this family. If $l \leq tn$ for a given t, each binary choice function with the number of $x_1+ \ldots + x_i \geq i/t$, for $1 \leq i \leq tn$, is called binary choice function with t-dominating properties. There exist bijections between set of choice functions χ and sets of t-ary trees with n internal nodes in widely used representations. All t-dominating binary choice functions, with $l = tn$ and the number of $x_1+ \ldots + x_i = n$, are bit string representations of all t-ary trees of the set A [10]. This bit string representation is called x-sequence and also known as Zaks' sequence [19]. By t-dominating definition, in each subsequence $\{x_j\}_1^i$ $(1 \leq i \leq tn)$ the accumulated numbers of 1's is at least $\lceil i/t \rceil$. In other words, if this subsequence contains m 1's, then it contains at most $(t-1)\, m$ 0's.

For any given choice functions $\delta = <d_1, \ldots, d_l>$ and $\gamma= <g_1, \ldots, g_l>$, we say that δ and γ are in *decreasing lexicographical* order, if and only if there exists $i \in \{1, \ldots, l\}$ satisfying $d_i > g_i$ and $d_j = g_j$ for every $j < i$.

The x-sequence can be obtained directly from t-ary trees. Given a regular t-ary tree with n internal nodes, we label each internal node with 1 and each external node with 0. Reading tree labels in pre-order (recursively, visit first root and then all the sub trees from left to right), we get a bitstring with n 1's and $(t-1)$ $n+1$ 0's. As the last visited node is an external node, we omit the corresponding 0. For example, the x-sequence corresponding to the tree presented in Figure 1 is **x**=100101000.

Theorem. The following sets are in a 1-1 correspondence with each other [19]:

1) All the t-ary trees with n internal nodes,

2) All the 0-1 sequences $\{x_i\}_1^{tn}$ with n 1's and $(t-1)\, n$ 0' s having the t-dominating property.

In order to design an algorithm for generating the set of trees, an ordering is to be imposed; one of these ordering is B-order. This ordering is defined as follows [19].

Definition. Given two t-ary trees T and T', we say $T < T'$ in B-order if

1) T is empty and T' is not empty, or
2) T is not empty, and for some i $(1 \leq i \leq t)$
   a) $T_j = T'_j$ for $j = 1, 2, \ldots, i-1$, and
   b) $T_i < T'_i$ in B-order.

Our both generation algorithms in sequential and parallel, which are given in the next section, produce the 0-1 sequences in decreasing lexicographical order such that their corresponding t-ary trees are in reverse B-order.

3. Sequential Generation

In this section we give a new sequential generation algorithm for t-ary trees in reverse B-order with 0-1 encoding. The algorithm *GenX-Seq* given in Figure 2 generates 0-1 sequences corresponding to a t-ary tree. The algorithm produces the tree sequences by interchanging any adjacent 10 by 01, which causes a right shift in the corresponding 1. The generation sequence starts with the sequence $1^n 0^{(t-1)n}$. By the first possible interchange the nth 1 in the sequence is shifted one position to the right. By each right shift a new sequence is generated. The last generated code by the algorithm is $(10^{t-1})^n$. It is clear that any right shift in this sequence (last code) would violate the dominating property. With regard to the last code the nth 1 can be shifted $N = (t-1)\, n - t + 1$ bits to the right of the initial position, and consequently the $(n - 1)$ th 1 shifted $(t - 1)$ $(n - 1) - t+1$ bits and respectively the $(n - i)$ th $(0 \leq i < n)$ 1 can be shifted $(t - 1)\,(n - i) - t+1$ bits from the initial position. Consequently ith 1 can be shifted up to $t(i - 1) + 1$th bit in the sequence. Clearly the position of the first one is always unaltered.

The algorithm has two underlying recursions and initially is called with $X = 1^n\, 0^{(t-1)\, n}$, $k= N+1$, $l= n - 1$, and $q = 1$. The total required time for the generation of all the sequences is $O(C_{n,t})$, and easily can be proved to be in constant average time *O(1)* per sequence [2]. It should be noted that for obtaining a more efficiency the parameter X in the algorithm could be deleted and assumed to be a global variable. In this case after each recursion call in the algorithm, a reverse 0-1 interchange should be performed on X. Also we can convert the recursive algorithm to an iterative algorithm such that the algorithm generates the next sequence independently. In this case the time complexity of the algorithm in the worst case for generating one sequence from another is $O(n)$, and consequently complexity of generation all $C_{n,t}$ sequences would be equal to $O(n\,C_{n,t})$.

We now show a ranking algorithm, *i.e.* an algorithm, which gives the position of, a tree presented as 0-1 sequences in the reverse B-order list of sequences corresponding to a t-ary tree. For ranking algorithm we need to define the doubly indexed sequences G_l^k $(1 \le l \le k \le n)$ for a fixed value t as follows:

$$G_l^k = \begin{cases} 0 & l > \left\lfloor \dfrac{k-1}{t-1} \right\rfloor + 1, \\ 1 & 1 = 0, \\ G_{l-1}^k + G_l^{k-1} & \text{otherwise.} \end{cases}$$

Where G_l^k is the number of 0-1 sequences with $\left\lfloor \dfrac{k+t-1}{t-1} \right\rfloor$ ones that hold the dominating property and beginning with at least $\left\lfloor \dfrac{k+t-1}{t-1} \right\rfloor - l$ ones.

Lemma. The coefficients G_l^k verify the following relations [2]:

a) $G_{n-1}^{N+1} = C_{n,t}$,

b) $G_l^k = \sum_{j-0}^{l} G_j^{k-1}$, $j < k$ & $j < \left\lfloor \dfrac{k-1}{t-1} \right\rfloor + 1$.

With regard to the above Lemma and definitions of G_l^k, the ranking algorithm is given in Figure 3. The required time for the above algorithm depends on the length of sequence which is $O(tn)$.

The unranking algorithm essentially reverses the steps carried out in computing the rank. According to the rank of a tree, the position of its sequence in the reverse B-order list is specified. Therefore, this position is obtained by using G_l^k's $(1 \le k \le N+1,\ l \le 1 \le n)$. The unranking algorithm is given in Figure 4. Clearly the time complexity of the unranking algorithm is $O(tn)$.

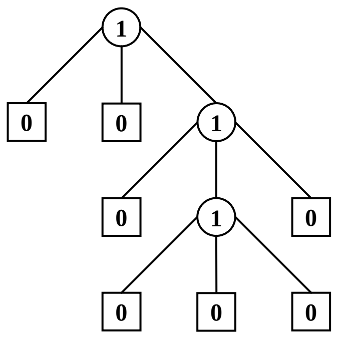

Figure 1. A 3-node 3-ary tree T, with encoding x={1, 0, 0, 1, 0, 1, 0, 0, 0}.

```
Procedure GenX-Seq (X: Xseq; k, l, q: Integer);
Begin
  If (k < N+1) Then Begin
    x_{N+n-k-l+1} := 0 ;
    x_{N+n-k-l+2} := 1 ;
  End;
  WriteXseq (X);
  If (k > l) Then Begin
    GenX-Seq (X, k-1, 1, l);
    If ( l < k ) And ( l < q ) And (l < ⌊(k-1)/(t-1)⌋ +1 ) Then
      GenX-Seq (X, k, l+1, q);
  End;
End;
```

Figure 2. X-sequences generation algorithm in the reverse order of B-order.

```
Function Rank (X: Xseq): Integer;
Var r, k, l, i: Integer;
Begin
  k := N+1 ; l := n - 1 ;
  i: = 1 ; r := 0 ;
  While (i ≤ t × n) and (l ≥ 0) Do Begin
    If (x_i = 0) Then Begin
      r := r + G_l^k ;
      k := k - 1 ;
    End
    Else
      l := l - 1 ;
    i := i + 1 ;
  End;
  Rank: = r + 1;
End;
```

Figure 3. Rank algorithm.

4. Parallel Generation Algorithm

In this section we present a parallel algorithm that generates 0-1 sequences of the form $\{x_1, x_2, ..., x_{tn}\}$ as defined in Section 2. The algorithm generates tree sequences in reverse B-order. Our algorithm is cost-optimal and adaptive and is executed on CREW SM SIMD model with *d* processors. As it is mentioned in the Section 2 the generation sequence starts with the sequence $1^n\ 0^{(t-1)\ n}$ and the last generated sequence is $(1\ 0^{t-1})^n$. The algorithm produces the next possible generated codes by the right interchanging of the 1's in the initial string. Each 1 can be shifted up to a specific position.

The algorithm uses four arrays *Y, Z, W* of length *n* and *X* of length *tn*, in shared memory. The *i*th elements of these arrays are denoted by y_i, z_i, w_i and x_i, respectively. Array *X* is simply an output buffer where any new sequence generated is placed, and initial value of it is $1^n\ 0^{(t-1)\ n}$. The other three arrays are used to store intermediate results, and are defined in the below.

1) Array *Y* holds the limit position for shifting of each 1, and is set to

$$y_i = t\,(i - 1) + 2\ ,\quad 1 \le i \le n.$$

2) Array *Z* hold the position of 1's in the generated code, and initially is set to

$$z_i = i\ ,\quad 1 \le i \le n.$$

3) Array *W* keeps track of those 1's that have reached their limiting position.

$$w_i = \begin{cases} True & \text{if } z_i = y_i, \\ False & \text{otherwise,} \end{cases}$$

and initially w_i $(1 \le i \le n)$ is false.

The algorithm given in Figure 5. It uses the processors $p_1, \dots, p_d$. The arrays *Y, Z, W* are subdivided to *d* subsequences of length $\lceil n/d \rceil$ and assigned to each processor. As it is mentioned, the algorithm produce the next possible generated code by the interchanging a 1 that the position is kept in array *Z*. This position is equal to any *i* such that $w_{i-1} = False$ and $w_i = True$. For any sequences unique position will have this property, therefore only one of the processor can obtain this position and keeps the index of this position in variable *k*. Employing the array *Z* and variable *k*, the corresponding 1 is right interchanged and the next sequence is generated, and then array *W* and *Z* are updated. Because of concurrent access of processors to *k* and z_k, therefore our computational model should be CREW.

```
Function Unrank (r: Integer): Xseq;
Var X: Xseq; i, j: Integer;
Begin
  For i: = 1  To t × n Do
    $x_i$ := 0 ;
  i := 1 ; l := n - 1 ; k:=N + 1 ;
  While ($i \le t{\times}n$) And ($l \ge 0$) Do Begin
    If ($r > S_l^k$) Then Begin
      $x_i$:= 0 ;
      r := r - $S_l^k$ ;
      k := k - 1 ;
    End
    Else Begin
      $x_i$:= 1 ;
      l := l - 1 ;
    End;
    i := i + 1 ;
  End;
  Unrank: = X;
End;
```

Figure 4. Unrank algorithm.

```
Procedure Parallel-GenX-Seq;
Var i, j, k: Integer; Flag: Boolean;
Begin
  WriteXseq (X);
  While ($w_1$ = False) Do Begin
    k: = n ;
    For i: = 1  To d Do In Parallel
      For j: = (i-1) × $\lceil n/d \rceil$ + 1 To i × $\lceil n/d \rceil$ Do
        If ($w_{j-1}$ = False) And ($w_j$ = True) And (j > 1) And
        ($j \le n$) Then
          k := j-1 ;
    End;
    $x_{z_k}$ : = 0 ; $z_k := z_k + 1$ ; $x_{z_k}$ : = 1 ;
    For i: = 1  To d Do In Parallel
      For j: = (i-1) × $\lceil n/d \rceil$ + 1  To i × $\lceil n/d \rceil$ Do
        If ($j \ge k$) And ($j \le n$) Then Begin
          $x_{z_j}$ := 0 ;
          $z_j := z_k + (j - k)$ ;
          $x_{z_j}$ := 1 ;
        End;
    End;
    If ($z_k < y_k$) Then
      WriteXseq (X);
    For i: = 1  To d Do In Parallel
      For j: = (i – 1) × $\lceil n/d \rceil$ + 1 To i × $\lceil n/d \rceil$ Do
        If ($j \le n$) Then
          If ($z_j = y_j$) Then
            $w_j$: = True
          Else
            $w_j$ := False ;
    End;
  End;
End;
```

Figure 5. Parallel version of *GenX-Seq* algorithm.

The time complexity of the algorithm depends on the existing loops. All the parallel loops in the algorithm performed in $O(n/d)$. The main loop for generating all the t-ary trees with n nodes is performed $C_{n,t}$ times. Thus, the overall time complexity of algorithm is $T(n) = O(nC_{n,t} / d)$. Since the algorithm is employed d processors, therefore its cost is $C(n) = O(nC_{n,t})$, and with regard to the time complexity of discussed sequential algorithm *Next*, it is cost-optimal.

Our algorithm can be also implied to an EREW model without violating the cost-optimality. If our model alters to EREW, concurrent read can be simulated by broadcasting the values of k and z_k on $O(log\ d)$. Therefore, the complexity of the algorithm would be equal to $T(\text{n}) =O(\ (n / d + log\ \ d)\ C_{n,t}$, and the cost is $C(n) = O((n\ + d\ \ log\ \ d)\ C_{n,t})$, which for $\text{d} \le n / log\ \ n$ is cost-optimal.

5. Conclusion

A Parallel generation algorithm for t-ary trees is presented. This algorithm is the second parallel algorithm for generation of t-ary trees in reverse B-order with 0-1 sequences. The algorithm is executed on a SM SIMD model. Both CREW and EREW model can support the cost-optimality of the algorithm with different number of processors. The algorithm is adaptive and the number of processors is variable and can be less than the number of internal nodes in a t-ary tree.

The authors have implemented and tested the algorithm in MPI on a 8 nodes Linux cluster system, and for different values of n the cost-optimality is obtained experimentally.

References

1. Ahrabian H. and Nowzari-Dalini A. Generation of *t*-ary trees with Ballot sequences. *Intern. J. Comput. Math.*, **80**: 1243-1249 (2003).
2. Ahrabian H. and Nowzari-Dalini A. On the generation of binary trees, from (0-1) codes. *Ibid.*, **69**: 243-251 (1998).
3. Ahrabian H. and Nowzari-Dalini A. On the generation of P-sequences. *Adv. Modeling Optim.*, **5**: 27-38 (2003).
4. Akl S.G. *The Design and Analysis of Parallel Algorithms*. Prentice Hall, Englewood Cliffs (1989).
5. Akl S.G. and Stojmenovic I. Generating t-ary trees in parallel. *Nordic J. Comput.*, **3**: 63-71 (1996).
6. Er M.C. Efficient generation of k-ary trees in natural order. *Comput. J.*, **35**(3): 306-308 (1992).
7. Kaprlski A. New methods for the generation of permutations, combinations and other combinatorial objects in parallel. *J. Parallel Distrib. Comput.*, **17**: 315-329 (1993).
8. Korsh J.F. A-order generation of k-ary trees with 4k-4 letter alphabet. *J. Infom. Optim. Sci.*, **16**(3): 557-567 (1995).
9. Kokosinski Z. On the generation of permutations through decomposition of symmetric group into osets. *Bit*, **30**: 583-591 (1990).
10. Kokosinski Z. On parallel generation of t-ary trees in an associative model. *Lecture Notes in Computer Science*, **2328**: 228-235 (2002).
11. Mirsky L. *Transversal Theory*. Academic Press, Washington (1971).
12. Pallo J. Generating trees with n nodes and m leaves. *Intern. J. Comput. Math.*, **21**: 133-144 (1987).
13. Roelants Van Baronaigien D. and Ruskey F.Generating t-ary trees in A-order. *Inform. Process. Lett.*, **27**(4): 205-213 (1988).
14. Ruskey F. Generating t-ary trees lexicographically. *SIAM J. Comput.*, **7**(4): 424-439 (1978).
15. Trojanowski E. Ranking and listing algorithm for k-ary trees. *SIAM J. Comput.*, **7**(4): 492-509 (1978).
16. Vajnovszki V. and Phillips C. Optimal parallel algorithm for generating k-ary trees. In: Woodfill M.C. (Ed.), *Proc. 12th International Conference on Computer and Applications*, ISCA, Raleigh, 201-204 (1997).
17. Vajnovszki V. and Phillips C. Systolic generation of k-ary trees, *Parallel Process. Lett.*, **9**(1): 93-101 (1999).
18. Yongjin Z. and Jianfang W. Generating k-ary trees in lexicographic order. *Sci. Sin.*, **23**: 1219-1225 (1980).
19. Zaks S. Lexicographic generation of ordered tree. *Theoret. Comput. Sci.*, **10**: 63-82 (1980).