AMO - Advanced Modeling and Optimization, Volume 8*, Number* 1*,* 2006

Adaptive Generation of t-ary Trees in Parallel [1]

H. Ahrabian And A. Nowzari-Dalini
School of Mathematics, Statistics, and Computer Science,
University of Tehran, Tehran, Iran.
And
Institute for Studies in Theoretical Physics and Mathematics (I.P.M.),
Tehran, Iran.
E-mail: {ahrabian,nowzari}@ut.ac.ir

Abstract

In this paper, a new adaptive and cost-optimal parallel algorithm is presented for generating t-ary tree sequences in reverse B-order. The algorithm generates t-ary trees by z-sequences in lexicographical order. Computations run in an EREW SM SIMD model. Prior to the discussion of the parallel algorithm a new sequential generation algorithm with $O(1)$ average time complexity is also given.

Keywords*:* t-ary Trees, Parallel Algorithm, B-order, z-Sequences, Recursion.

1 Introduction

Trees, especially t-ary trees have been extensively studied and many algorithms have been given to generate all n nodes t-ary trees [2, 3, 6, 9, 11, 13, 14, 15, 18, 19]. In some of these algorithms trees are represented by integer sequences and the corresponding sequences are generated. Some of well-known encodings are p-sequences [5, 11], inversion table [8], 0-1 sequences [19]. Any generation algorithm imposes an ordering on the set of trees. One of the well-known ordering on trees is B-order [19], and many integer sequences corresponding to t-ary trees are generated in B-order [12, 14, 19].

Recently, in order to increase the speed of the generation, few parallel algorithms for generation of t-ary trees have been published for various parallel models [4]. The advantage of a parallel generating algorithm over an equivalent version is that t-ary trees may be generated more efficiently with low cost. Well-known parallel generating algorithms for t-ary trees are those of Stojmonovic and Akl [5], Vajonovszki and Phillips [16, 17], and

[1]This research was in part supported by a grant from I.P.M. (No. CS1384-4-07).

Kokosinski [10]. In the first of them [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 processors. In the second [16], trees are represented by 0-1 sequences and the algorithm is run on a CREW 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 model. The latter two algorithms generate sequences of length tn with tn processors. Finally, Kokosinski [10] generated t-ary trees of n nodes by 0-1 sequences in parallel with an associative model with n processors.

Adaptive Generation of t-ary Trees in Parallel In this paper, we present a parallel generation algorithm for t-ary trees. Here, t-ary trees are represented with 0-1 sequences and generated in lexicographical order such that the corresponding trees are in reverse B-order. These 0-1 sequences are introduced by Zaks and are called z-sequences[19]. The computational model for our parallel algorithm is an EREW SM SIMD model. The algorithm is adaptive and cost-optimal. A parallel algorithm is said to be adaptive if it is capable of modifying its behavior according to the number of processors actually available on the parallel computer being used. An algorithm is cost-optimal if the number of processors it uses multiplied by its running time matches up to a constant factor, a lower bound on the number of operations required to solve the problem sequentially. This property can be further specified to the way in which the lower bound is defined [4]. Prior to the discussion of our adaptive parallel algorithm a new sequential generation algorithm is also given. Consecutive sequences are generated, by this sequential algorithm, in lexicographic order with $O(n)$ time complexity in worst case and $O(1)$ in constant average time.

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

2. Definitions

We assume the reader to be familiar with the definition of t-ary tree, extended t-ary tree, internal node, leaf, subtree, sibling and descendent as given in [7]. The number of internal nodes in a t-ary tree T is denoted by $|T|$. The number of t-ary trees with n internal nodes is denoted by $C_{n,t}$ and is known to have the following value: $C_{n,t} = \frac{1}{(t-1)n+1}\binom{tn}{n}$.

In order to encode a t-ary tree T with n internal nodes, we label each internal node with 1 and each leaf with 0, then we read these labels in preorder (recursively, visit first root and then all the subtrees from left to right), we get a sequence with n 1's and $(t-1)n+1$ 0's. As the last visited node is always a leaf, we omit the corresponding 0 and

show this sequence by $f(T) = \{x_i\}_1^{tn} = x$. From x we build a sequence $h(x) = \{z_i\}_1^n = z$ such that z_i is position of ith 1 in x. See Figure 1 for an example. The sequence x has t-dominating property if in each subsequence $\{x_i\}_1^{\ell}(1 \leq \ell \leq tn)$ the accumulated numbers of 1's is at least $\lceil \ell/t \rceil$. Clearly, the number of 1's in each prefix is at least equal to the number of 0's and the number of 0's in each suffix is at least equal to the number of 1's. The dominating property for the corresponding z-sequence is defined as $z_i \leq (i-1) \times t+1$ for $i = 1, 2, \cdots, n$.

The following theorem establishes the relation between t-ary trees and integer sequences [19].

Theorem 1 *The following sets are in 1-1 correspondence with one another,*

1. *all the t-ary trees with n internal nodes,*
2. *all the integer sequences $\{z_i\}_1^n$ such that $0 < z_1 < z_2 < \cdots < z_n$ and $z_i \leq (i-1) \times t+1$ for $i = 1, 2, \cdots, n$.*

As it is mentioned, 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].

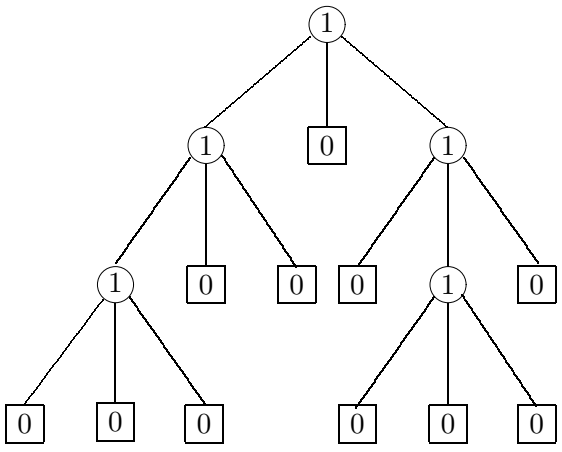

Figure 1. A 5-node 3-ary tree T, with encoding $z = \{1, 2, 3, 10, 12\}$.

Definition 1 *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 $(1 \leq i \leq t)$, we have*
   a) *$T_j = T'_j$ for $j = 1, 2, \cdots, i-1$, and*
   b) *$T_i < T'_i$.*

The following theorem illustrates the relation of the orders between z-sequences and t-ary trees [19].

Theorem 2 *Let T and T' be two t-ary trees with n internal nodes, and let z and z' sequences be defined as above. The following relations are equivalent:*

1. *$T < T'$ in B-order,*
2. *$z > z'$.*

Considering the above theorem, our both generation algorithms in sequential and parallel, which are given in the next sections, produce the z-sequences in lexicographical order such that their corresponding t-ary trees are in reverse B-order.

3. Sequential Generation

In this section, a new sequential generation algorithm with z-sequences is given. The algorithm *GenZ-Seq* given in Figure 2 generates z-sequences corresponding to a t-ary trees. The algorithm produces the tree sequences by incrementation. The generation sequence starts with the sequence $\{1, 2, 3, \cdots, n\}$. By each incrementation a new sequence is generated. The last generated code by the algorithm is $\{1, 1t+1, 2t+1, 3t+1, \cdots, (n-1)t+1\}$. It is clear that any incrementation in this sequence (last code) would violate the dominating property. With regard to the last code the nth element can be incremented up to $(n-1)t$, and consequently the $(n-1)$th element incremented up to $(n-2)t$, and respectively the $(n-i)$th $(0 \leq i < n)$ element can be incremented up to $(n-i-1)t$. Clearly the value of the first one is always unaltered.

The algorithm has two underlying recursions and initially is called with $Z = \{1, 2, 3, \cdots, n\}$, $k = N + 1$, $\ell = n - 1$, and $q = 1$, where $N = (n-1) \times t - n + 2$ is the difference between nth element of last code with ℓ, and is used to control the number of generation in each recursion call. Also, in this algorithm z_i denotes the ith element of the array Z.

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 [1]. It should be noted

```
Procedure GenZ-Seq (Z : Zseq;  k,  ℓ,  q : Integer ) ;
Begin
    If ( k < N + 1 ) Then
        z_{n−ℓ+1} := z_{n−ℓ+1} + 1 ;
    WriteZseq ( Z ) ;
    If ( k > 1 ) Then Begin
        GenZ-Seq (Z,  k − 1,  1,  ℓ ) ;
        If ( ℓ < k ) And ( ℓ < q ) And (ℓ < ⌊(k−1)/(t−1)⌋ + 1 ) Then
            GenZ-Seq (Z,  k,  ℓ + 1,  q ) ;
    End ;
End ;
```

Figure 2. Sequential z-sequences generation algorithm.

that for obtaining a more efficiency, the parameter Z in the algorithm can be deleted and assumed to be a global variable. In this case after each recursion call in the algorithm, a decrementation should be performed on Z.

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 $O(nC_{n,t})$.

4. Parallel Generation Algorithm

In this section, we present a parallel algorithm that generates z-sequences of the form $\{z_1, z_2, \cdots, z_n\}$ as defined in Section 3. Our algorithm is cost-optimal and adaptive and is executed on an EREW SM SIMD model with d processors. As it is mentioned in the Section 2, the generation sequence starts with the sequence $\{1, 2, 3, \cdots, n\}$ and the last generated sequence is $\{1, 1t+1, 2t+1, \cdots (n-1)t+1\}$. The algorithm produces the next possible generated codes by the incrementation in the initial sequence. Each element can be incremented up to a specific value.

The algorithm uses three arrays Z, Y and W of length n, in shared memory. The ith elements of these arrays are denoted by z_i, y_i and w_i, respectively. These arrays are used to store intermediate results, and are defined in the below.

1) Array Z is simply an output buffer where any new sequence generated is placed, and initially is set to

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

2) Array Y holds the upperbound value incrementation of each element, and is set to

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

3) Array W keeps status of elements in Z 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 $(1 \leq i \leq n)$ is *False*.

Our parallel algorithm is given in Figure 3, and uses the processors $p_1, \cdots, p_d$. The arrays Z, Y and 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 inceremented the value of an element in the array Z. The position of this element is equal to any i such that $w_{i-1} = False$ and $w_i = True$ and its value be less than y_i. For any sequence a unique position will have this property, therefore only one of the processors can obtain this position and keeps the index of this position in variable k and is broadcast to all processors. Employing the variable k, the z_k is incremented and the next sequence is generated, and then array W is updated such that if z_k is equal to y_k then w_k takes *True* value, in other case it takes *False*. Then *ParallelMove* searches in parallel in W for *True* values between two consecutive *False* values, if such values exist then they are set to *False* values in parallel. This determines the position of next element for incrementing and generating the sequence in the next step. The algorithm *ParallelMove* is illustrated in Figure 4.

The time complexity of the algorithm depends on the existing loops. The main loop for generating all the t-ary trees with n internal nodes is performed $C_{n,t}$ times, and all the parallel loops in the algorithm performed in $O(n/d)$. The time complexity of *BroadCast* and *ParallelMove* is $O(\log d)$. Thus, the time complexity of the algorithm for generating a code from another in worst case is $T(n) = O(n/d + \log d)$.

Theorem 3 *The presented Parallel-GenZ-Seq is cost-optimal and adaptive.*

Proof. Since the algorithm employs d processors, therefore its cost is $C(n) = O(n + d \log d)$. With respect to the discussion in previous section the time complexity of sequential generation algorithm in worst case is $O(n)$ and therefore the parallel algorithm is cost-optimal for $d \leq n/log\ n$. Since the number of processors employed in the algorithm is not a fixed number, therefore the adaptivity of algorithm is obvious. ■

```
Procedure Parallel-GenZ-Seq ;
Var  i, j, k : Integer ;
Begin
   WriteZseq (Z) ; k := n ;
   While (k > 1) Do Begin
      BroadCast (k) ;
      If (z_k < y_k) Then Begin
         z_k := z_k + 1 ;
         For i := 1 To  d Do In Parallel
            For j := (i − 1) × ⌈n/d⌉ + 1 To  i × ⌈n/d⌉ Do
               If (j > k) And (j ≤ n) Then
                  z_j := z_k + (j − k) ;
         End ;
         WriteZseq (Z) ; k := n ;
      End ;
      Else Begin
         For i := 1 To  d Do In Parallel
            For j := (i − 1) × ⌈n/d⌉ + 1 To i × ⌈n/d⌉ Do
               If (j ≤ n) Then Begin
                  If (z_j = y_j) Then
                     w_j := True
                  Else
                     w_j := False ;
               End ;
         End ;
         ParallelMove (W) ;
         k := 1 ;
         For i := 1 To  d Do In Parallel
            For j := (i − 1) × ⌈n/d⌉ + 1 To  i × ⌈n/d⌉ Do
               If (j > 1) And (j ≤ n) And (w_{j−1} = False) And (w_j = True) Then
                  k := j ;
         End ;
      End ;
   End ;
End ;
```

Figure 3. Parallel version of *GenZ-Seq* algorithm.

```
Procedure ParallelMove (var W : BooleanArray) ;
Var i, j : Integer ; V : BooleanArray ;
Begin
    For i := 1 To  d Do In Parallel
      For j := i × ⌈n/d⌉ DownTo  (i − 1) × ⌈n/d⌉ + 1 Do
        If (j < n) And (w_j = True) Then
          w_j := w_{j+1} ;
    End ;
    For i := 1 To d Do In Parallel
      v_i := v_{(i−1)×⌈n/d⌉+1} ;
    End ;
    For i := 1 To d Do In Parallel
      For j := 0 To ⌈log d⌉ − 1 Do
        If (i + 2^j ≤ d) And (v_i = True) Then
          v_i := v_{i+2^j} ;
    End ;
    For i := 2 To d Do In Parallel
      If w_{(i−1)×⌈n/d⌉} = True Then
        w_{(i−1)×⌈n/d⌉} := v_i ;
    End ;
    For i := 1 To d Do In Parallel
      For j := i × ⌈n/d⌉ DownTo  (i − 1) × ⌈n/d⌉ + 1 Do
        If (j < n) And (w_j = True) Then
          w_j := w_{j+1} ;
    End ;
End ;
```

Figure 4. Parallel move algorithm.

5. Conclusion

We have given a simple parallel algorithm for generating t-ary trees with n internal nodes by z-sequences. The algorithm is the first parallel algorithm for generation of t-ary trees in reverse B-order on an EREW SM SIMD model. As it is mentioned, the previous given parallel algorithms for this purpose are designed on linear array, CREW SM SIMD and associative memory model. Our model can support the cost-optimality of the algorithm with different number of processors. The algorithm is adaptive and the number of processors are variable and can be less than the number of internal nodes in a t-ary tree, and produces z-sequences in lexicographical order.

References

[1] H. Ahrabian and A. Nowzari-Dalini, Generation of t-ary trees with Ballot sequences, *Intern. J. Comput. Math.*, **80** (2003), 1243–1249.

[2] H. Ahrabian and A. Nowzari-Dalini, Gray code algorithm for listing k-ary trees, *Studies in Informatics and Control*, **13** (2004), 243–251.

[3] H. Ahrabian and A. Nowzari-Dalini, On the generation of P-sequences, *Adv. Modeling Optim.*, **5** (2003), 27–38.

[4] S.G. Akl, *The design and analysis of parallel algorithms*, Prentice Hall, Englewood Cliffs, 1989.

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

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

[7] D. E. Knuth, *The art of computer programming, Vol. 1: Fundemental algorithms*, 2nd. Ed., Addison-Wesley, Reading, 1973.

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

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

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

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

[12] J. Pallo and R. Racca, A note on generating binary tree in A-order and B-order, *Intern. J. Comput. Math.*, **18** (1985), 27-39.

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

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

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

[16] V. Vajnovszki and C. Phillips, Optimal parallel algorithm for generating k-ary trees, *in Proc. 12th International Conference on Computer and Applications*, (ed. M. C. Woodfill), ISCA, Raleigh, 1997, 201–204.

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

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

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