Parallel Generation of P-sequences

Ahrabian H., Nowzari-Dalini A.
Center of Excellence in Biomathematics, School of Mathematics, Statistics, and Computer Science, University of Tehran, Tehran, Iran
[*] *Corresponding author, e-mail: ahrabian@ut.ac.ir*

(*received: 13/1/2007 ; accepted: 28/12/2007*)

Abstract

We present a cost-optimal and adaptive parallel algorithm for generating t-ary trees with P-sequences. The computational model employed in this algorithm is an exclusive read exclusive write with a shared memory single instruction multiple data computer. Our parallel algorithm is the first designed P-sequence generation on this model. Prior to the discussion of this parallel algorithm, a new sequential algorithm for generation of t-ary trees with P-sequences in $O(1)$ constant average time per sequence is presented.

Keywords: B-order, Parallel Algorithms, P-sequences, t-ary Trees.

1. Introduction

The generation of certain interesting combinatorial objects plays a significant role in computer science. If we wish to produce an exhaustive list of combinatorial objects, the time required to create the representation of two consecutive objects in the list is extremely important since the cardinality of a class is generally an exponential function of the size of generated objects. In order to increase the speed of the generation, many parallel algorithms for certain combinatorial objects have recently been published (Ahrabian and Nowzari 2007, 2005, Akl et al. 1990, 1994, 1996, 1996b, Stojmenovic 1990). The advantage of a parallel generating algorithm over an equivalent sequential version is that objects may be generated with constant delay. The model of parallel computation should be as simple as possible, and the algorithms be cost-optimal and adaptive. A generating algorithm satisfying these criteria is optimal in every suitable sense. There are parallel algorithms which satisfy these optimality criteria for generating combinatorial objects such as permutations (Akl et al. 1994), combinations (Akl et al. 1990), derangements (Akl et al. 1992), subsets and equivalence relations (Stojmenovic 1990), binary and t-ary trees (Akl et al. 1996, Kokosinski 2002, Vajnovszki 1996), variations (Akl et al. 1991, Stojmenovic 1996), and well-formed parentheses strings (Akl et al. 1996).

Trees are one of the important combinatorial objects in computer science and have many applications. A list of all t-ary trees might be used to search for a counter example to some conjectures, or to test and analyze an algorithm for its correctness or complexity. Usually, trees are encoded as integer sequences and instead of generations of trees, these sequences can be generated. Some of well-known encodings are P-sequences (Akl et al. 1996, Pallo 1987), inversion tables (Kontt 1977), 0-1 sequences (Zaks 1980). This paper deals with the exhaustive generation of P-sequences for encoding trees. Few clever sequential algorithms for generating lists of P-sequence are discussed in (Gupta 1991, Pallo 1987). These algorithms have constant amortized time, i.e., the total amount of computation divided by the number of generated objects is constant, but linear in the worst case.

Some papers dealing with the parallel generation of t-ary trees with different encodings have been published (Ahrabian and Nowzari 2007,2005, Akl et al. 1996, Kokosinski 2002, Vajnovszki et al. 1997, 1999). Akl and Stojmenovic (1996) represented trees by an inversion table and the employed processor model is a linear array multiprocessor. The generated integer sequences corresponding to the t-ary trees of n internal nodes in this algorithm are of length n and the parallel algorithm is executed by n processors. Vajonvszki and Phillips (1997) represented trees by 0-1 sequences and the algorithm is run on a shared memory multiprocessor. Vajonvszki and Phillips (1999) presented a parallel generating algorithm for t-ary trees represented by P-sequences on a linear array, which is the only parallel algorithm for P-sequences.

The latter two algorithms generate sequences of length *tn* with *tn* processors. Kokosinski (2002) generated t-ary trees of n internal nodes by 0-1 sequences in parallel with an associative model by *n* processors. Ahrabian and Nowzari (2005) presented an adaptive and cost-optimal parallel generation algorithm for t-ary tree sequences by 0-1 sequences in an Exclusive Read Exclusive Write (EREW) Shared Memory (SM) Single Instruction Multiple Data (SIMD) model (see Akl (1989) for more details about this model).

In this work we describe two new sequential and parallel algorithms for generating t-ary trees with P-sequences. Our sequential algorithm generates each sequence in constant average time $O(1)$. Algorithm satisfies the properties given in the previous section, and is adaptive and cost-optimal. The employed computational model is an EREW SM SIMD computer, and the algorithm is the first designed P-sequence generation technique on this model. It should be noted that EREW SM SIMD is the simplest and most implementable model in parallel computing.

The rest of this paper is organized as follows. The notations and definitions required for further sections are given in Section 2. New sequential generation algorithm for P-sequences is discussed in Section 3. In Section 4, the corresponding parallel algorithm is presented. Finally, the conclusion is given in Section 5.

2. Notations and Definitions

A t-ary tree T with n internal nodes can be defined recursively as being either an external node or an internal node together with a sequence $T_1, T_2, ..., T_t$ of t-ary trees. T_i is defined as the ith subtree of T. An n internal nodes t-ary tree has $(t - 1)n + 1$ external nodes. The set of t-ary trees with n internal nodes is denoted by $T_{n,t}$ and the cardinality of this set, denoted by $C_{n,t}$, is known to be

$$C_{n,t} = \frac{1}{(t-1)n+1}\binom{tn}{n}.$$

P-sequence were initially introduced in (Pallo 1987) for coding and generation binary trees. These results are generalized for t-ary trees in (Pallo 1987). As it is mentioned earlier, the parallel algorithm given in this paper generates the t-ary trees represented by generalized P-sequences, defined as follows.

Definition 1. Given a tree $T \in T_{n,t}$, the P-sequence of T is the integer sequence $p_1 p_2 ... p_{n(t-1)}$ where p_i is the number of internal nodes written before leaf i in pre-order traversal of T. The integer corresponding to the last leaf, the $n(t - 1) + 1$th leaf, is always equal to n, therefore we can omit this number.

For example the P-sequence corresponding to the t-ary tree T denoted in Fig. 1 is the integer sequence p = 233333334455555.

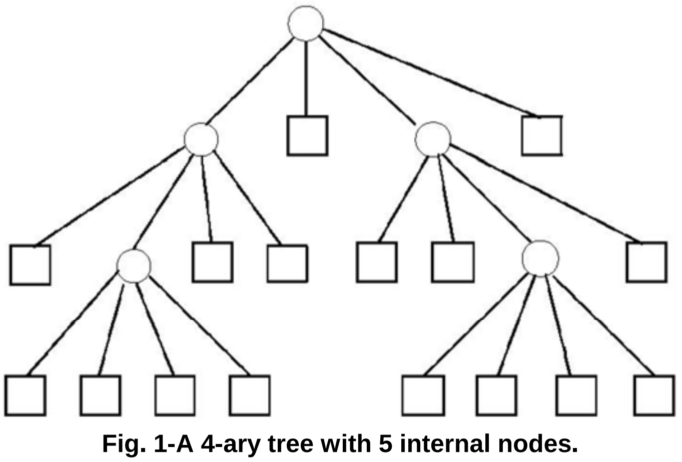
Fig. 1-A 4-ary tree with 5 internal nodes.

Lemma. (Pallo 1987) An integer sequence $p_1 p_2 ... p_{n(t-1)}$ is the P-sequence of a tree T in $T_{n,t}$ if and only if:

$1) p_i \le p_{i+1} \quad for \quad all\, i, \quad 1 \le i \le n(t-1) - 1,$

$2) P_{n(t-1)} = n,$

$3) 1 + \left\lfloor \frac{i-1}{t-1} \right\rfloor \le p_i \quad for \quad all \quad i, \quad 1 \le i \le n(t-1),$

where $\lfloor x \rfloor$ is the largest integer less than or equal to x.

Let $P_{n,t}$ be the set of P-sequences for coding n internal nodes t-ary trees. The first P-sequence, in decreasing lexicographic order, in $P_{n,t}$ is

$$p = \underbrace{n \ldots n}_{(t-1)n},$$

and the last one is

$$p = \underbrace{1 \ldots 1}_{t-1}\underbrace{2 \ldots 2}_{t-1} \ldots \underbrace{n \ldots n}_{t-1}.$$

Any generation algorithm imposes an ordering on the set of trees. Such an ordering is called B-order (Zaks 1980).

Definition 2. Given two t-ary trees T and T', we say $T < T'$ in B-order if:
1) T is leaf and T' is not leaf, or
2) For some i (1 ≤ i ≤ t),
 a) $T_j = T'_j$ *for j =1, 2,...,i − 1,* and
 b) $T_i < T'_i$ in B-order.

In (Pallo 1987) it is shown that the lexicographic order over $P_{n,t}$ matches the B-order of trees in $T_{n,t}$. In this paper, P-sequences are generated in decreasing lexicographic order and their correspondding trees are in the reverse B-order.

3. Sequential Generation Algorithm

In this section, we describe an algorithm for the generation of P-sequences corresponding to t-ary trees. Giving a sequence as input, the next sequence in $P_{n,t}$ is generated by the algorithm *Next* given in Fig. 2. This algorithm generates consecutive P-sequences in decreasing lexicographical order which their corresponding trees are in the reverse B-order.

Recall from Section 1, a P-sequence of a t-ary tree T is an integer sequence $p = p_1 p_2 \ldots p_{(t-1)n+1}$, where p_i is the number of visited internal nodes during visiting the *i*th external node in the preorder traversal of T. The algorithm produces each P-sequence by decrementing an element of the input code. As we can see in the algorithm, the first loop, searches for the first *decreaseable element* from right to left in the sequence. The *decreaseable element* is an element whose value is greater than the corresponding value in the last generateable P-sequence in $P_{n,t}$. In other words, employing P-sequence properties given in the lemma in Section 2, this *decreaseable element* is *i*th element in the sequence whose value differs from its predecessor element and is greater than $1 + \left\lfloor \frac{i-1}{t-1} \right\rfloor$. After determining the position of this element, its value is decremented, and the elements on the right are set to n. Clearly, to generate all the set $P_{n,t}$, algorithm *Next* is called $C_{n,t}$ times, while the initial input sequence is equal to the first P-sequence and each generated sequence would be the input of the next iteration.

```
Function Next ( p : Pseq ): Pseq ;
    Var  i,  j : Integer ;
    Begin
        i :=(t-1)n ;
        While (i ≥ 1) and ( (p_i = p_{i-1}) Or (p_i = (1+ (i-1)  Div  (t-1))) ) Do
            i :=i-1 ;
        If (i>0) And (p_i<>(1+(i-1)  Div  (t-1))) Then Begin
            p_i := p_i − 1;
            For j :=i+1 To (t-1)n  Do
                p_j := n;
        End ;
        Next :=p ;
    End ;
```

Fig. 2-The algorithm Next for generating P-sequences.

Regarding the steps of this algorithm, its time complexity in worse-case is $O(tn)$. Therefore, the total required time for generating all P-sequences in worst-case is $O(tnC_{n.t})$. With a discussion similar to the one given in (Pallo 1987), we can prove that the average time complexity of this algorithm is $O(t)$.

4. Parallel Generation Algorithm

In this section a new parallel algorithm for the generation of P-sequences is presented. The computational model for this algorithm, illustrated in Fig. 3, is EREW SM SIMD with $N \leq (t-1)n$ processors. The principal idea is to let each processor generate a part of the P-sequence. The sequence p is subdivided into N subsequences of length $m = \lceil (t-1)n/N \rceil$ and the processor i is assigned $\{p_{(i-1)m+1}, p_{(i-1)m+2},\ldots,p_{im}\}$. All the processors now perform the following processes: Each processor i finds the position of rightmost *decreaseable element* and stores its position in f_i, then between all the N computed values f_i, the maximum position is evaluated. Let ℓ be this maximum position. This value is broadcasted to the all processors. The above process requires $O(tn/N) + O(\log N)$ where $O(tn/N)$ is the time complexity for finding the *decreaseable element*, and $O(\log N)$ is the time required for finding the maximum value of N elements in a shared memory and broadcasting. Later for each processor i ($\lceil \ell/m \rceil \leq i \leq N$) we let p_j=n where $(i-1)m+1 \leq j \leq im$. The recent operations requires $O(tn/N)$ time, therefore the total required time for this algorithm is $T(n) = O(tn/N + \log N)$.

Theorem. The *Parallel-Next* algorithm is cost-optimal and adaptive.

Proof. Regarding the time complexity of *Parallel-Next*, the cost of this algorithm is equal to $C(n) = O(tn + N \log N)$.

By considering the time complexity of the sequential algorithm *Next* in worst case, given in previous section, this parallel algorithm is cost-optimal for $N \leq (tn) / \log(tn)$ processors. The adaptivity of this algorithm is clear.

```
Procedure Parallel-Next ( Var p : Pseq );
Var  i, j, l, v : Integer ; F : Array [1..N] of Integer ;
Begin
  For i := 1 To N Do In Parallel
    f_i :=0 ;
    For j := im DownTo (i − 1)m +1 Do
      If ( j ≤ (t − 1)(n − 1) ) And ( p_j > p_j−1 ) And
         (p_j > ⌊(j − 1)/(t − 1)⌋ +1 ) And ( f_i =0) Then
         f_i :=j ;
End ;
For i := 1 To N Do In Parallel
  For j := 0 To ⌈log N ⌉− 1 Do
    If (2i +2^j − 1 ≤ N ) And ((i − 1) Mod 2^j = 0) And
      (f_{2i-1} < f_{2i+2^j-1}) Then
      f_{2i-1} := f_{2i+2^j −1} ;
End ;
ℓ := f_1 ;
 p_ℓ := p_ℓ −1;
For i := ⌈ℓ/m⌉ To N  Do In Parallel
  For j := im DownTo (i − 1)m +1 Do
    If ( j ≥ ℓ ) And ( j ≤ (t − 1)n ) Then
       P_j := n ;
    End ;
  End ;
```

Fig. 3-Parallel version of the algorithm Next.

5. Conclusion

We have developed a parallel algorithm for generating n internal nodes t-ary trees represented by P-sequences. This algorithm is a parallelized version of a novel sequential generation algorithm presented in this paper. The algorithm is adaptive and cost-optimal, and satisfy all the desirable properties of a parallel algorithm. The computational model employed in this algorithm is an EREW SM SIMD model. This model is the simplest and most implementable model in parallel computing. Clearly, any designed parallel algorithm in this model can be implemented on the other models. Therefore, our parallel algorithm can be implemented on any parallel computer.

References

Ahrabian, H. Nowzari-Dalini, A. 2007: Parallel generation of t-ary trees in A- order. *Comput. J.* **50:** 581-588.

Ahrabian, H. Nowzari-Dalini, A. 2005: Parallel generation of t-ary trees. *J. Sci .I R. Iran* **16:** 169-173.

Akl, S.G. 1989: The Design and Analysis of Parallel Algorithms. Prentice Hall, Englewood Cliffs.

Akl, S.G., Clavert, J. and Stojmenovic, I. 1992: Systolic generation of derangements. In: Algorithms and Parallel VLSI Architectures II (eds. P. Quinton and Y. Robert), *Elseivier, New York*, P. 49-70.

Akl, S.G., Duboux, T. Stojmenovic, I. 1991: Constant delay parallel counters. *Parallel Process. Lett.* **1:** 143–148.

Akl, S.G., Gries, D. Stojmenovic, I. 1990: An optimal parallel algorithm for generating combination. Inform. *Process. Lett***.** **33:** 135–139.

Akl, S.G., Meijer, H. Stojmenovic, I. 1994: An optimal systolic algorithm for generating permutations in lexicographic order. *J. Parallel Distrib. Comput.* **20:** 84–91

Akl, S.G. Stojmenovic, I. 1996: Generating combinatorial objects on a linear array of processors. in: Parallel Computing: Paradigms and Applications (ed. A. Y. Zomaya), *International Thomson Computer Press, Boston,* p. 639–670.

Akl, S.G. Stojmenovic, I. 1996 b: Generating t-ary trees in parallel. *Nordic J. Comput.* **3:** 63-71.

Gupta, D.K. 1991: On the generation of P-sequences. *Intern. J. Comput. Math.* **38:** 31– 35.

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

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

Pallo, J. 1987: Generating trees with n nodes and m leaves. Intern. J. Comput. Math. **21:**133-144

Pallo, J. Racca, R. 1985: A note on generating binary trees in A-order and B- order. *Intern. J. Comput. Math.* **18:** 27-39.

Stojmenovic, I. 1990: An optimal algorithm for generating equivalence relation on a linear array of processors. *Bit* **30:** 424–436.

Stojmenovic, I. 1996: Generating n-ary reflected Gray codes on a linear array of processors, *Parallel Process. Lett.* **6:** 27–34.

Vajnovszki, V. Phillips, C. 1997: Optimal parallel algorithm for generating k-ary trees. in: Proc. 12th International Conference on Computer and Applications (ed. M. C. Woodfill), *International Society for Computers and their Applications, Raleigh,* p.201-204.

Vajnovszki, V. Phillips, C. 1999: Systolic generation of k-ary trees. *Parallel Process. Lett.* 9: 93-101.

Zaks, S. 1980: Lexicographic generation of ordered trees. *Theoret. Comput. Sci.* **10:** 63-82.