Parallel Solution of Recurrences 
on a Tree Machine

Roy P. Pargas

Received June 1983; revised July 1984

The recurrence

\[ x_0 = a_0 \]
\[ x_i = a_i + b_i x_{i-1}, \quad i = 1, 2, \ldots, n - 1 \]

requires \( O(n) \) operations on a sequential computer. Elegant parallel solutions exist, however, that reduce the complexity to \( O(\log N) \) using \( N \geq n \) processors. This paper discusses one such solution, designed for a tree-structured network of processors.

A tree structure is ideal for solving recurrences. It takes exactly one sweep up and down the tree to solve any of several classes of recurrences, thus guaranteeing a solution in \( O(\log N) \) time for a tree with \( N \geq n \) leaf nodes. If \( n \) exceeds \( N \), the algorithm efficiently pipelines the operation and solves the recurrence in \( O(n/N + \log N) \) time.

**KEY WORDS:** Tree machine; parallel computation; recurrences.

1. **INTRODUCTION**

Consider the first-order linear recurrence

\[ x_0 = a_0 \]
\[ x_i = a_i + b_i x_{i-1}, \quad i = 1, 2, \ldots, n - 1 \]

where \( n \geq 1 \) represents the number of terms of the recurrence and \( a_i \) and \( b_i \) are real scalars. The solution of Eq. 1, i.e., the set of values \( (x_0, x_1, x_2, \ldots, x_{n-1}) \), is obtained in a straightforward manner by a sequen-
tial algorithm requiring a total of \( n - 1 \) multiplications and \( n - 1 \) additions, i.e., \( O(n) \) operations, and little can be done on a sequential computer to improve the algorithm complexity. Elegant parallel solutions exist, however, that reduce the complexity to \( O(\log N) \) using \( N \geq n \) processors. This paper discusses one such solution, designed for a tree-structured network of processors.

Solving recurrences quickly is important because recurrences are so often components of larger problems. A tridiagonal linear system of equations, for example, can be transformed into several recurrence problems. Solving the recurrences provides a solution to the linear system.

A tree structure is ideal for solving recurrences. It takes exactly one sweep up and down the tree to solve any of several classes of recurrences, thus guaranteeing a solution in \( O(\log N) \) time for a tree with \( N \geq n \) leaf nodes. If \( n \) exceeds \( N \), the algorithm efficiently pipelines the operation and solves the recurrence in \( O(n/N + \log N) \) time.

This paper has four major parts. A description of the tree machine model used is presented in Section 2. The general tree algorithm, RECUR, is described and proven correct in Section 3. Recurrences to which RECUR is applicable are described in Section 4. These include first-, second- and higher-order linear recurrences, and recurrences of the form:

\[
x_0 = a_0
\]

\[
x_i = (a_i + b_i x_{i-1})/(c_i + d_i x_{i-1}) \quad i = 1, 2, ..., n-1
\]

Extensions and variations of RECUR are presented in Section 5. Finally, conclusions and general remarks are given in Section 6.

Parallel solutions of linear recurrences have been studied before. In a paper on the parallel solution of tridiagonal linear systems, Stone\(^{(1)}\) introduced a method called recursive doubling, which allows one to solve linear recurrences of all orders in \( O(\log N) \) steps on a parallel processor of the ILLIAC-IV type. The method was generalized by Kogge and Stone\(^{(2)}\) and by Kogge.\(^{(3)}\) They described a broad class of functions that enjoy special composition properties and to which the method is applicable. Kogge\(^{(4)}\) also described how to pipeline the method to obtain the maximal computational rate.

Studies on the relationship between computation time and number of processors when solving recurrences\(^{(5\text{-10})}\) have resulted in bounds on the number of processors required to minimize the time to solve first-order linear recurrences and bounds on the time required to solve the problem given a fixed number of processors. Except for the algorithm described by Gajski,\(^{(9)}\) the algorithms were designed for an idealized \( N \)-processor machine on which there is no contention for memory (to obtain either