

Binary Trees and Parallel Scheduling Algorithms

ELIEZER DEKEL AND SARTAJ SAHNI

Abstract—This paper examines the use of binary trees in the design of efficient parallel algorithms. Using binary trees, we develop efficient algorithms for several scheduling problems. The shared memory model for parallel computation is used. Our success in using binary trees for parallel computations, indicates that the binary tree is an important and useful design tool for parallel algorithms.

Index Terms—Complexity, design methodologies, parallel algorithms, scheduling, shared memory model.

I. INTRODUCTION

Algorithm design techniques for single processor computers have been extensively studied. For example Horowitz and Sahni [15] extoll the virtues of such design methods as divide-and-conquer, dynamic programming, greedy method, backtracking, and branch-and-bound. These methods generally lead to efficient sequential (i.e., single processor) algorithms for a variety of problems. These algorithms, however, are not very efficient for computers with a very large number of processors. In this paper, we propose a design method that we have found useful in the design of algorithms for computers that have many processors. The method proposed here is called the *binary tree method*. While this method has been used in the design of parallel algorithms earlier, here we attempt to show its broad applicability to the design of such algorithms. It is hoped that further research will bring to light some other basic design tools for parallel algorithms. One should note that trees have been used extensively in the design of efficient sequential algorithms. In fact, divide-and-conquer, backtracking, and branch-and-bound all use an underlying computation tree [15]. The use of binary trees as proposed here is quite different from the use of trees in sequential computation.

With the continuing dramatic decline in the cost of hardware, it is becoming feasible to build computers with thousands of processors economically. In fact, Batcher [5] describes a computer (MPP) with 16 384 processors that is currently being built for NASA. In coming years, one can expect to see computers with a hundred thousand or even a million processing elements. This expectation has motivated the study of parallel algorithms. Since the complexity of a parallel algorithm depends very much on the architecture of the parallel computer on which it is run, it is necessary to keep the architecture in mind when designing the algorithm. Several parallel architectures have been proposed and studied. In this paper, we shall deal directly only with the single instruction stream, multiple data stream (SIMD) model. Our technique and algorithms readily adapt to the other models (e.g., multiple instruction stream multiple data stream (MIMD) and data flow models). SIMD computers have the following characteristics:

- 1) They consist of p processing elements (PE's). The PE's are indexed $0, 1, \dots, p-1$ and an individual PE may be referenced as in PE(i). Each PE is capable of performing the standard arithmetic and logical operations. In addition, each PE knows its index.
- 2) Each PE has some local memory.
- 3) The PE's are synchronized and operate under the control of a single instruction stream.
- 4) An enable/disable mask can be used to select a subset of the PE's that are to perform an instruction. Only the enabled PE's will

Manuscript received October 4, 1980; revised July 13, 1982. This work was supported in part by the National Science Foundation under Grant MCS80-005856 and in part by the Office of Naval Research under Contract N000145-80-C-0650.

E. Dekel is with the Program in Computer Science, University of Texas at Dallas, Richardson, TX 75080.

S. Sahni is with the University of Minnesota, Minneapolis, MN 55455.

perform the instruction. The remaining PE's will be idle. All enabled PE's execute the same instruction. The set of enabled PE's can change from instruction to instruction.

While several SIMD models have been proposed and studied, in this paper we shall be primarily concerned with only the shared memory model (SMM). In the shared memory model, there is a common memory available to each PE. Data may be transmitted from PE(i) to PE(j) by simply having PE(i) write the data into the common memory and then letting PE(j) read it. Thus, in this model it takes only $O(1)$ time for one PE to communicate with another PE. Two PE's are not permitted to write into the same word of common memory simultaneously. The PE's may or may not be allowed to simultaneously read the same word of common memory. If the former is the case, then we shall say that *read conflicts* are permitted. In a realistic situation, the common memory will be divided into blocks of size q and a read conflict occurs whenever two PE's attempt to simultaneously access the same block. Throughout this paper, we assume that $q = 1$.

Most algorithmic studies of parallel computation have been based on the SMM. Agerwala and Lint [1], Arjomandi [2], Csanky [8], Eckstein [11], and Hirschberg [12] have developed algorithms for certain matrix and graph problems using the SMM. Hirschberg [13], Muller and Preparata [24], and Preparata [30] have considered the sorting problem for SMM. The evaluation of polynomials on the SMM has been studied by Munro and Paterson [25], while arithmetic expression evaluation has been considered by Brent [7] and others.

The mesh connected computer (MCC), cube connected computer (CCC), and perfect shuffle connected computer (PSC) are three other SIMD models. Efficient algorithms to sort and perform data permutations on an MCC can be found in Thompson and Kung [38], Nassimi and Sahni [26] and [27], and Thompson [37]. Thompson's algorithms [37] can also be used to perform permutations on a CCC and a PSC. Lang [19], and Lang and Stone [20], and Stone [36] show how certain permutations may be performed using shuffles and exchanges. Nassimi and Sahni [28] develop fast sorting and permutation algorithms for a CCC and PSC. Dekel, Nassimi, and Sahni [9] present efficient matrix multiplication and graph algorithms for CCC's and PSC's.

The algorithms considered in this paper are described explicitly only for the SMM. The algorithms are readily translated into algorithms for the other SIMD models. In some cases, it may be necessary to use the data broadcasting algorithms developed by Nassimi and Sahni [29] to accomplish this adaptation to the other models.

Throughout this paper, we assume that no read conflicts are allowed.

II. THE BINARY TREE METHOD

In the binary tree method, we make use of a binary computation tree. Such a tree is generally a complete binary tree. Fig. 1 shows such a tree with 11 leaf nodes. The nodes of this tree have been indexed (indexes appear outside each node) using the standard indexing scheme for complete binary trees.

With each node of the computation tree, we associate k , $k \geq 1$ subproblems. The computation consists of k passes over this computation tree. In pass 1, we proceed from the leaves to the root solving the first subproblem associated with each node; in pass 2, we proceed from the root to the leaves solving the second subproblem associated with each node; and so on. Every odd pass is from the leaves to the root while every even pass is from the root to the leaves.

As far as we are aware, the binary tree method has thus far been used only with $k = 1$. A simple example is finding the sum $\sum_{i=1}^n A(i)$ of n numbers. The computation tree used has n leaf nodes. The subproblem associated with each node is that of finding the sum of all the numbers represented by the leaves in the subtree of which it is a root. Fig. 1 shows the computation for the case $n = 11$.

