Parallel algorithms for finding the minimum spanning tree of a weighted undirected graph and the bridge-connected and biconnected components of an undirected graph on a linear array of processors are presented. On an n-vertex graph, our algorithms perform in $O(n^2/p)$ time on an array of size $p$, for all $p$, $1 \leq p \leq n$, thus providing optimal speedup for dense graphs. The paper describes two approaches to limit the communication requirements for solving the problems. The first is a divide-and-conquer strategy applied to Sollin's algorithm for finding the minimum spanning tree of a graph. The second uses a novel data-reduction technique that constructs an auxiliary graph with no more than $2n-2$ edges, whose bridges and articulation points are the bridges and articulation points of the original graph.

1. Introduction

In this paper we present parallel algorithms for a number of graph problems on a fixed-size network of processors. The processors are connected together in the form of a one-dimensional array so that any processor (except the two at the end) can directly communicate with only its left and right neighbors. Each processor in the array is a sequential random access machine [1] equipped with its own local random-access memory. Communication between adjacent processors in the array may be effected either by means of bidirectional communication links between them or through a common memory shared by two adjacent processors.

The problems that we consider in this paper are those of finding the connected components, the minimum spanning tree of a weighted graph, and the bridge-connected and biconnected components of an undirected graph. On a graph of $n$ vertices represented by an $n \times n$ adjacency matrix, we solve all of the above mentioned problems in $O(n^2/p)$ time using a linear array of $p$ processors, for all $p$, $1 \leq p \leq n$. Thus our algorithms achieve a speedup linear in the number of processors employed within the range 1 to $n$. In particular, with $n$ processors, the algorithms require an optimal $O(n)$ time. Informally, since $O(n^2)$ data items of the adjacency matrix must be examined by the algorithm, at least $O(n^2/p)$ time is needed, and, since computations on this model require $O(p)$ time to communicate over the array diameter, $\Omega(n)$ can be shown to be a lower bound on the time complexity for this model. Our results compare favorably with previously described algorithms for these problems on fixed-size processor networks (see Section 1.2). In addition, the simplicity of the computational model makes it attractive from a practical viewpoint.

Our algorithm for finding the minimum spanning tree (MST), which obviously may also be used for determining the connected components, is based on a "divide-and-conquer" strategy applied to Sollin's [2] algorithm for the problem. Our algorithms for finding the bridges and the articulation points of a graph are quite new, in that they are not implementations of known sequential or parallel algorithms that, to our knowledge, have appeared in the literature. The algorithms for the latter two problems employ a novel method of data reduction to construct an auxiliary graph composed of at most $2n-2$ edges, whose articulation points (bridges) are also the articulation points (bridges) of the original graph.

In the next section we describe in more detail the computation model employed. In section 1.2, we review the known parallel algorithms for these problems and indicate briefly the difficulty in attempting to directly adapt
these schemes to this model. In sections 2, 3 and 4 we present the algorithms for finding the minimum spanning tree, the bridges and the articulation points respectively. For convenience we have assumed that the input to our algorithms is a connected graph; the modifications required to accommodate disconnected graphs involve purely clerical details.

1.1. Computation Model

The computation model employed is a linear array of \( p \) processors. Each processor has an index from the set \( \{0, \ldots, p-1\} \), and the processor \( j, 1 \leq j \leq p-2 \), is connected by bidirectional communication links to the processors \( j-1 \) and \( j+1 \), which are referred to as the left and the right neighbors respectively of \( j \). A processor may communicate only with its left or right neighbor in a unit time step.

Each processor in the array is a sequential random access machine (RAM) equipped with a sizable amount of local random-access memory. As in the uniform-cost sequential RAM model [1], each access by a processor to its local memory for either reading or writing requires unit time. Operands are limited to be of size \( O(\log n) \) bits, where \( n \) is the size of the input. This restriction applies both to the number of bits that may be accessed from memory in one cycle and to the number of bits that may be communicated between neighbors in unit time.

Algorithm performance on this model can be compared with the sequential time complexities for the problem on the sequential uniform cost random access machines (See [1]).

1.2. Overview

Several sequential algorithms for computing the minimum spanning tree (forest) of an undirected graph are known; among them are those by Prim[3], Dijkstra[4], Kruskal[5], Sollin[2], and Cheriton and Tarjan[6]. Among the parallel algorithms proposed on fixed-connection networks of processors, are those proposed by Bentley[7], Bentley and Ottmann[8], Leighton[9], Nath, Maheshwari and Bhatt[10], Atallah and Kosaraju[11], and Huang[12]. Bentley [7] and Bentley and Ottmann [8] described algorithms for computing the MST on a set of \( n \) points on a binary tree and a linear array \( \sqrt{n} \) processors respectively in \( O(n \log n) \) time. In the case that \( p, p < n/\log n \), processors are available, the algorithm performs in time \( O(n^2/p) \), thus achieving linear speedup for all \( p \) in the range \( 1 \leq p \leq n/\log n \). Leighton [9], Nath, Maheshwari and Bhatt [10] provided \( O(\log^3 n) \) time solutions for the problem on a \( n \times n \) mesh of trees network. Atallah and Kosaraju [11] presented an \( O(n) \) time algorithm for this problem using an \( n \times n \) mesh connected array of processors. Finally, Huang [12] described an algorithm that achieves a time complexity of \( O(n^2/p) \) for all \( p, 1 \leq p \leq \sqrt{n/\log n} \), on a \( \sqrt{n} \times \sqrt{p} \) mesh of trees network.

With the exception of the algorithms proposed in [7-9] the algorithms in the above list are not efficient for implementation on a network having a fixed number of processors. The algorithm presented in this paper requires time \( O(n^2/p) \) using a linear array of \( p \) processors, for all \( p, 1 \leq p \leq n \). The challenge in designing the algorithm arises from the need to limit both the total amount of data communication required as well as the distance through which data has to be communicated. The peculiar problems arising in this context do not manifest themselves in the high bandwidth networks such as meshes of trees [9], which in addition to having several parallel data paths also possess a much smaller (log \( n \)) communication diameter. On the other hand, previous algorithms [7,8], on limited bandwidth models like ours, yield linear speedup only up to \( n/\log n \) processors.

Parallel algorithms on PRAM models for finding the biconnected components of a graph include those of Savage and Ja-Ja[13], Tsin and Chin[14] and Tarjan and Vishkin[15]. Huang’s algorithm [12] for finding the biconnected components on a mesh of trees is based on the algorithm proposed in [14,15]. It is, however, difficult to efficiently employ the same technique on a linear array of processors. The difficulty arises from the amount of data communication the algorithm demands, and, the distance over which the data must be transmitted in every iteration of the algorithm. A simulation of this technique on a linear array, for example, requires time \( O(n^2/p + (\sqrt{p} + p)\log n) \). In this case linear speedup is achieved only for \( p \) less than \( (n/\log n)^{p^2} \). In the above time-complexity expression, the term \( n\sqrt{p} \log n \) arises from \( \log n \) iterations, in each of which, some link in the array must, in the worst case, handle up to \( O(n^{\sqrt{p}}) \) amount of data from each of \( p \) processors. The last term, \( p \log n \) arises from the consideration of the diameter over which the communication occurs. Clearly, a scheme that must perform efficiently on a linear array over a wider range of \( p \) must address the possibility of reducing the data, and the communication necessary to solve the problem.