Abstract

We show how to test outerplanarity in time $T(n) = O(\log n \log \log n)$ using $n/T(n)$ processors of CREW PRAM. It is the first optimal parallel algorithm recognizing a nontrivial class of graphs and it is the main result of the paper. If the graph is outerplanar and biconnected then a Hamiltonian cycle is produced. Using this cycle and optimal parsing algorithm for bracket expressions the construction of the tree of faces as well as vertex colourings (with the smallest number of colours) are also done by optimal parallel algorithms.

1. Introduction

A parallel algorithm working in time $T(n)$ with $P(n)$ processors is optimal iff the product $T(n)P(n)$ is linear. We are interested in NC algorithms (with polylogarithmic $T(n)$ and polynomial $P(n)$). The list of such optimal algorithms in graph theory is very short. Mostly optimal algorithms in this area solve problems on trees. A notable exception is a series of parallel algorithms on planar graphs (connected and biconnected components, spanning trees, 5-colouring), see [H] and [HCD].

In this paper we add to the list of optimal parallel algorithms in graph theory several algorithms on outerplanar graphs: testing, embedding, optimal vertex colouring.

Outerplanar graphs form a subclass of planar graphs. A planar graph is outerplanar if there is an embedding in the plane such that all its nodes lie on the same outer (infinite) face. Such embedding will be called outerplanar. Outerplanar graphs are well suited to parallel computations because of a tree
structure of their faces. It is known that the set of faces of the outerplanar graph is the same for all possible outerplanar embeddings. Assume that we exclude from considerations the outer infinite face and by a face we mean any other face. Consider a graph whose nodes are faces and in which there is a connection between each two faces with a common edge. Then this graph is a forest. Assume that the outerplanar graph is biconnected. In this case the graph of faces is a tree. We call it the tree of faces. Now it is not surprising that many problems have optimal parallel algorithms for outerplanar graphs, because computations on trees are easy. However we have first to compute such a tree and then we have to design the algorithms in such a way that they reflect the tree structure of the outerplanar graph. (See [S] for more on outerplanar graphs.)

Our model of the computations is a CREW PRAM, see [GR2] for the definition.

2. Optimal Parallel Algorithm For Outerplanarity Testing

In this section we construct first an almost optimal parallel algorithm: $T(n)=O(\log n \log^* n)$, $P(n)=O(n)$. In the outerplanar graph the number of edges is bounded by $2n-3$, where $n$ is the number of nodes (the size of the graph). We can check whether this is true for the input graph and assume later that the number of edges is $O(n)$. We can assume also that the input graph is biconnected, because Hagerup's algorithm for finding biconnected components [H] can be applied to the input graph and then one can test outerplanarity for each component independently. A biconnected outerplanar graph is an $n$-gon with edges classified as sides and diagonals.

A standard representation of an undirected graph is used. The graph is given by a set of doubly linked adjacency lists. Each edge $(u,v)$ on the adjacency list of $u$ also points to the edge $(v,u)$ on the adjacency list of $v$.

Let $K>80$ be a constant. The node of the graph is called small iff its degree is at most $K$, otherwise it is big. The simple cycle $C$ is short iff its length is at most $K$ and there is no shorter cycle consisting of some nodes of $C$. We define the concepts of reducible nodes and cycles.

Each node of degree one is reducible. The node of greater degree is reducible iff it is of degree 2 and one of its neighbours is a small node.

The cycle is reducible iff it is a short cycle and contains at most one big node. For each reducible cycle $C$ we distinguish one small node of $C$ (later associated with $C$) and one of the edges incident to the big node if $C$ contains a big node.

The algorithm is based on the following lemmas: