Coarse Grained Parallel Graph Planarity Testing[*]

Edson Cáceres
Dept. de Computação e Estatística
Univ. Federal do Mato Grosso do Sul
Campo Grande, MS, Brazil

Albert Chan
School of Computer Science
Carleton University
Ottawa, Canada

Frank Dehne
School of Computer Science
Carleton University
Ottawa, Canada

S. W. Song
Inst. de Matem. e Estatística
Universidade de São Paulo
São Paulo, SP, Brazil

Abstract *We present a coarse grained parallel algorithm for planarity testing and planar embedding. The algorithm requires $O(\log^2 p)$ communication rounds and linear sequential work per round. It assumes that the local memory per processor, N/p, is larger than p^ϵ for some fixed $\epsilon > 0$. This assumption is true for all commercially available multiprocessors. Our result implies a BSP algorithm with $O(\log^2 p)$ supersteps, $O(g \log^2(p) \frac{N}{p})$ communication, and $O(\log^2(p) \frac{N}{p})$ local computation. Our algorithm is based on the general structure of a previous PRAM method presented by Klein, using a parallel implementation of PQ-trees. The main contribution of this paper lies in the study of many of the individual PRAM steps that are very inefficient on existing commercial parallel machines. We present non-trivial, efficient, CGM implementations of the various parts of the overall strategy proposed by Klein. Our main result is a parallel planarity testing algorithm which is much more practical and efficient on commercially available multiprocessors.*

Keywords: Parallel algorithms, coarse-grained multicomputer, graph planarity, BSP

[*]Research partially supported by the Natural Sciences and Engineering Research Council of Canada, FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo), CNPq, the CNPq/NSF Collaborative Research Program, FINEP and Pronex.

1 Introduction

In this paper, we consider the problem of detecting whether a graph G is planar and, if so, reporting a planar embedding of G.

The first sequential linear-time algorithm for planarity testing and planar embedding was presented by Hopcroft and Tarjan [7]. Booth and Lueker [1] described another linear time algorithm which is based on a previous version by Lempel, Even and Cederbaum [12] but uses *PQ*-trees and *st*-numbering. JáJá and Simon [8] showed that the problem is in NC. Using [1, 12], Klein and Reif [11] presented an efficient PRAM algorithm which requires $O(\log^2 n)$ time using a linear number of processors. Using open ear decomposition and *st*-numbering, Ramachandran and Reif [15] presented a PRAM algorithm with linear work.

Unfortunately, theoretical results from PRAM algorithms do not necessarily match the speedups observed on *real* parallel machines. In this paper, we present a parallel planarity testing and planar embedding algorithm that is based on a more practical parallel model. More precisely, we will use a version of the BSP model [16] referred to as the *Coarse Grained Multicomputer* (CGM) model. In comparison to the BSP model, the CGM [4] allows only bulk messages in order to minimize message overhead. A CGM is comprised of a set of p processors $P_1, \ldots, P_p$ with

$O(N/p)$ local memory per processor and an arbitrary communication network (or shared memory). All algorithms consist of alternating local computation and global communication rounds. Each communication round consists of routing a single h-relation with $h = O(N/p)$, i.e. each processor sends $O(N/p)$ data and receives $O(N/p)$ data. We require that all information sent from a given processor to another processor in one communication round is packed into one long message, thereby minimizing the message overhead. A CGM computation/communication round corresponds to a BSP superstep with communication cost gN/p (plus the above "packing requirement"). Note that every CGM algorithm is also a BSP algorithm but not vice versa. Finding an optimal algorithm in the coarse grained multicomputer model is equivalent to minimizing the number of communication rounds as well as the total local computation time. The CGM model has the advantage of producing results which correspond much better to the actual performance on commercially available parallel machines.

In this paper, we present a parallel CGM algorithm for planarity testing and planar embedding. The algorithm requires $O(\log^2 p)$ communication rounds and linear sequential work per round. It assumes that the local memory per processor, N/p, is larger than p^ϵ for some fixed $\epsilon > 0$. This assumption is true for all commercially available multiprocessors. Our result implies a BSP algorithm with $O(\log^2 p)$ supersteps, $O(g \log^2(p)\frac{N}{p})$ communication, and $O(\log^2(p)\frac{N}{p})$ local computation. Our algorithm is based on the general structure of the PRAM method in [9], using a parallel implementation of PQ-trees. However, many of the PRAM steps in [9] are very inefficient on existing commercial parallel machines, making the overall method in [9] impractical. The main contribution of this paper lies in the study of these individual steps and how they can be implemented on a CGM. We present non-trivial, efficient, CGM implementations of the various parts of the overall strategy in [9]. Our main result is a parallel planarity testing algorithm which is much more practical and efficient on commercially available multiprocessors.

2 Coarse-Grained Parallel Graph ST-Numbering

Before we proceed with our planarity testing algorithm, we require a parallel st-numbering method for the CGM. An st-numbering of a graph G with vertices 1, 2, ..., n is a numbering of the vertices where the following is verified. Vertices 1 and n are adjacent and each j of the other vertices is adjacent to two vertices i and k such that $i < j < k$. Figure 1 shows an example (from [14]) of a graph and an st-numbering.

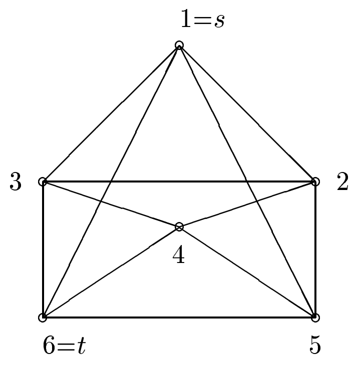

Figure 1: An st-numbered Graph

We observe that a parallel st-numbering method can be obtained by combining the CGM graph algorithms of [3] and the PRAM algorithm by Maon, Schieber and Vishkin [13]. We now give a brief outline of the method.

Let $G = (V, E)$ be a graph with an open ear decomposition $G = [P_0, P_1, \ldots, P_l]$ such that $P_0 = (s,t)$ and the endpoints $L(P_i)$ and $R(P_i)$ of each P_i are in P_j and P_k where $i > j \geq k \geq 0$. The vertex in an ear P_i, $i \geq 1$, which is adjacent to $L(P_i)$ (respectively $R(P_i)$), is denoted by $LS(P_i)$ (respectively $RS(P_i)$). The vertex $L(P_i)$ is called the *anchor* of P_i and if $v \in P_i$, $v \neq L(P_i)$ and $v \neq R(P_i)$, then v is called an *internal* vertex.

Algorithm 1 Computing the st-Numbering of a Graph [13]

Input: An open ear decomposition of graph $G = (V, E)$, $G = [P_0, P_1, \dots, P_l]$, where $P_0 = (s, t)$.

Output: A valid st-numbering of graph G.

```
(1) Compute the ear tree $ET(V_T, E_T)$ where
    $V_T = \{P_i | P_i$ is an ear in $G\}$ and $E_T = \{(P_i, P_j) |\ L(P_i)$ is an internal vertex of $P_j, i \geq 1\}$.
(2) Compute an orientation of the ears of $G$.
(3) Compute the $st$-numbering of the ears of $G$.
— End of Algorithm —
```

To implement Step 1, we apply the CGM open ear decomposition method in [3]. The adjacency lists of graph ET is constructed using CGM integer sorting [5]. Step 2 uses a *hinge tree.* Finding the hinges reduces to l lowest common ancestor computations in ET, using [3]. Step 3 uses a *numbering tree* NT and computation of a preorder numbering of NT. (Many of the details can be seen in [13].) The construction of NT can be done in $O(p)$ rounds. The preorder numbering can be done using using Euler Tour technique for computing tree functions [3].

Thus the CGM st-numbering algorithm can be done in $O(\log p)$ communication rounds and linear work per round since it is based on [3] ($O(\log p)$ rounds) and [5] (constant number of rounds) and both require linear work per round.

3 Coarse-Grained Parallel Graph Planarity Testing

In this section, we present the main result of this paper, a coarse-grained parallel algorithm for graph planarity testing. In the following, we first give an outline of the main structure of the algorithm and then detail how to implement the main steps on a CGM within the claimed time bounds.

Planarity testing using PQ-trees is very well described in [14]. The PRAM algorithms for planarity testing of [11] are also based on the PQ-tree data structure. Before we present our CGM algorithm, we give some idea.

A PQ-tree is a data structure suitable to represent a set of permutations on a ground set S [1]. Its internal nodes are called P-nodes and Q-nodes which are represented by circles and rectangles, respectively. The leaves are from the ground set S. The tree is ordered, i.e. the order of the children of a node is relevant. The children of a P-node can be reordered arbitrarily. The children of a Q-node can be reversed or flipped.

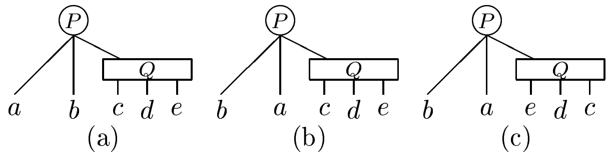

Figure 2: PQ-trees Representing Several Permutations of *abcde*

The order of the ground set in the PQ-tree, from left to right, is called the *frontier*, which is obviously a permutation of the ground set. Figure 2(a) shows a PQ-tree whose frontier is the permutation *abcde*. Figure 2(b) shows a reordering of the P-node giving the permutation *bacde*. If we perform also a Q-node flipping we get the permutation of Figure 2(c). Notice that in some permutations (for example (c)), the leaves a and e are consecutive.

By using the permissible operations on a PQ-tree T we can generate a set of permissible permutations, denoted by $L(T)$. A PQ-tree T' is equivalent to T if it can be transformed into T by a sequence of permissible transformations.

Given a set $A \subset S$, we say that $\lambda \in L(T)$ *satisfies* A if all elements of A occur consecutively in λ. Given a *reduction set* $\{A_1, \dots, A_k\}$ of subsets of S and a PQ-tree T, the *reduce* operation obtains a PQ-tree T' such that each permutation in $L(T')$ satisfies every A_i, $1 \leq i \leq k$.

We can now illustrate how st-numbering and

PQ-trees can be used in planarity testing.

Consider G an *st*-numbered graph of vertices in $V = \{1, 2, \dots, n\}$ (see example in Figure 1), and G_k the subgraph induced by the vertex set $V_k = \{1, \dots, k\}$. The *bush* graph B_k is formed by adding to G_k all edges, called virtual edges, with one end in V_k and the other in $V - V_k$. Such vertices in $V - V_k$ are called *virtual vertices* and are kept separate (so there may be several virtual vertices with the same label) and placed on the outer face, on a horizontal line.

B_k can be represented by a *PQ*-tree. A *P*-node represents a cut-vertex of B_k. A *Q*-node represents a bi-connected component of G_k. A leaf represents a virtual vertex of B_k. It can be shown [14] that if B_k is any bush graph of a planar graph G, then there exists a sequence of permutations and reversions so that all the vertices labeled $k + 1$ occupy consecutive positions on the horizontal line. Such permutations and reversions consist of repeated application of nine transformation rules called template matchings to the *PQ*-trees [1].

Consider again graph G of Figure 1. Consider B_k for $k = 3$, as shown in Figure 3(a). Figures 3(b) shows the corresponding *PQ*-tree. This tree is transformed so that all occurrences of vertex 4 (i.e. $k + 1$) are consecutive. This is possible for each B_k if G is planar. This constitutes the main idea of the sequential planarity algorithms [1, 12] which are based on vertex addition.

The parallel PRAM algorithm [11] and the proposed CGM algorithm use a divide-and-conquer scheme, by combining or joining already embedded subgraphs into larger ones. In the following description, we use the multiple disjoint reduction operation (MDReduce) on a *PQ*-tree [2]. It performs the *reduce* operation for a set $\{A_1, \dots, A_k\}$ of subsets of S, where the A_i are disjoint. We also use the ROTATE(A,B,C) operation to make $A \cup B$ and $B \cup C$ contiguous (see Figure 5).

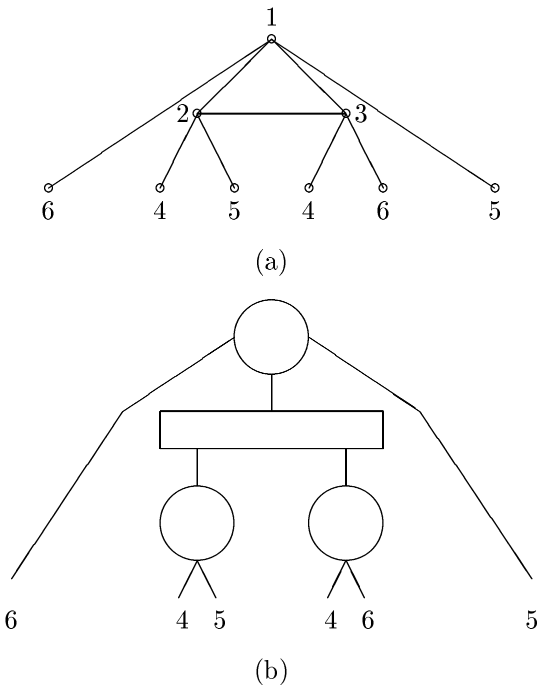

Figure 3: (a) B_3 and (b) the Corresponding *PQ*-tree where leaves 4 can be made consecutive

Algorithm 2 Graph Planarity Testing
Input: A graph G with n vertices and $m = O(n)$ edges.

Output: Test if G is planar and, if possible, report a planar embedding of G.

(1) Compute the bi-connected components of G as described in [3].

(2) Since a graph is planar if and only if all its bi-connected components are planar, we check each bi-connected components for planarity. Small components with less than $(\frac{n}{p})$ vertices can be stored in a single processor and can be tested by a sequential algorithm. For each larger components with $(n' \geq \frac{n}{p})$ vertices, a group of $p' = p\frac{n'}{n}$ processors is assigned to solve the problem. For the remainder, the algorithm only specifies the steps for *one* such group of processors that handles one bi-connected component. To simplify exposition, we let again n refer to the size of the component and p refer to the number of processor assigned to that component.

(3) Compute the *st*-numbering of the bi-

connected component as outlined in Section 2. Renumber all vertices according to the st-numbering with $s = 1$ and $t = n$.

(4) Let $G^{(0)} = G$. For $i = 1 \dots k = O(\log p)$ repeat:

(4.1) Select a set of joinable nodes and compute $G^{(i)}$ from $G^{(i-1)}$. See Algorithm 3 for details.

(4.2) Consider a set of joinable nodes; see Figure 4 for illustration and notation. Let $T(u_j)$ be the PQ-tree that is obtained from the previous iteration. From Klein and Reif [11], $B_j \neq \emptyset$ only for $j = 1$ and k. For each $j = 1 \cdots k$, if $B_j = \emptyset$, let $T'(u_j) = T(u_j)$. If $B_j \neq \emptyset$ (i.e. $j = 1$ or k), apply MDReduce [2] to $T(u_j)$ with respect to E_j and B_j, and a rotation $ROTATE(E_j, B_j, F_j)$ as illustrated in Figure 5. Call the resulting tree $T'(u_j)$. Finally for $k = 0$, apply MDReduce [2] to $T(u_0)$ with respect to $E_1, \cdots, E_k$ and call the resulting tree $T'(u_0)$.

(4.3) Join $T'(u_1), \cdots, T'(u_k)$ with $T'(u_0)$, see Algorithm 3 for details.

(4.4) Apply MDReduce [2] to $T'(v)$ with respect to $out(v)$. If $B_1 \neq \emptyset$, perform a $ROTATE(A, B_1, B_k \cup out(v))$. If $B_k \neq \emptyset$, perform a $ROTATE(A \cup B_1, B_k, out(v))$. We call the resulting tree $T(v)$.

(5) If any of the resulting PQ-trees is a null tree, STOP and report that G is not planar.

(6) If $G^{(k)}$ has more than $O(\frac{n}{p})$ edges then STOP and report that G is not planar.

(7) $G^{(k)}$ has $O(\frac{n}{p})$ vertices and $O(\frac{n}{p})$ edges. Compress $G^{(k)}$ into a single processor and apply a sequential planarity testing algorithm to $G^{(k)}$. G is planar if and only if $G^{(k)}$ is planar. A planar embedding of G can be derived from the planar embedding of $G^{(k)}$ and the PQ-trees derived in Step 4 by a reverse processing of Steps 4.1 and 4.2.

— End of Algorithm —

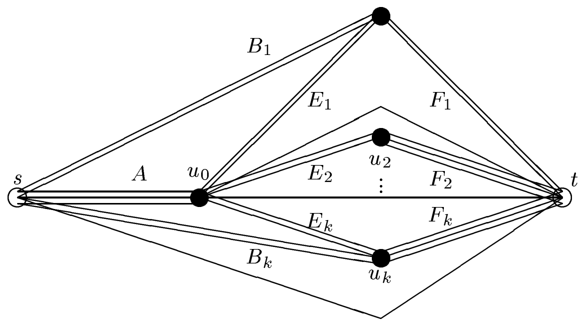

Figure 4: A Set of Joinable Nodes

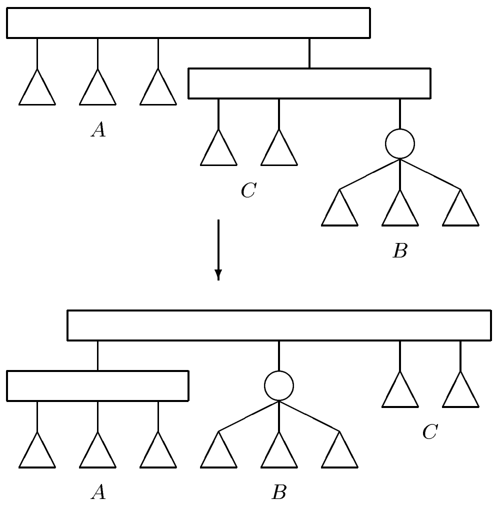

Figure 5: Illustration of Operation Rotate(A, B, C)

We now discuss the details of how to implement Step 4.1/4.3 on a CGM.

Algorithm 3 Details of Step 4.1/4.3 in Algorithm 2.

(1) if i mod $4 = 1$,

(1.1) For every nodes v other than s and t, define its parent $p(v)$ as follows: $p(v)$ is the highest st-numbered neighbor of v with st-number smaller than v. Using [3], compute the spanning tree

```
            $T^{(v-1)}$ of $G^{(v-1)}$ induced by the parent pointers and, for each node $v$, the distance of $v$ to the root $s$ of $T^{(v-1)}$. Identify each node as either *odd* or *even* based on its distance to $s$.
        (1.2) Select a set of joinable nodes as follows: Contract all *even* nodes of $T^{(v-1)}$ into their *odd* parents.
    (2) if $i$ mod $4 = 2$,
        (2.1) Re-use the spanning tree $T^{(v-2)}$ constructed in the previous stage (Note: We only need the even-odd labeling and parent pointers, and assume that they ware stored with the nodes and are available without re-computing the spanning tree.)
        (2.2) Select a set of joinable nodes as follows: Contract all *odd* nodes of $T^{(v-1)}$ into their *even* parents.
    (3) if $i$ mod $4 = 3$,
        (1.1) For every nodes $v$ other than $s$ and $t$, define its parent $p(v)$ as follows: $p(v)$ is the lowest $st$-numbered neighbor of $v$ with $st$-number larger than $v$. Using [3], compute the spanning tree $T^{(v-1)}$ of $G^{(v-1)}$ induced by the parent pointers and, for each node $v$, the distance of $v$ to the root $t$ of $T^{(v-1)}$. Identify each node as either *odd* or *even* based on its distance to $t$.
        (1.2) Select a set of joinable nodes as follows: Contract all *even* nodes of $T^{(v-1)}$ into their *odd* parents.
    (4) if $i$ mod $4 = 0$,
        (2.1) Re-use the spanning tree $T^{(v-2)}$ constructed in the previous stage (Note: We only need the even-odd labeling and parent pointers, and assume that they ware stored with the nodes and are available without re-computing the spanning tree.)
        (2.2) Select a set of joinable nodes as follows: Contract all *odd* nodes of $T^{(v-1)}$ into their *even* parents.
— End of Algorithm —
```

We now discuss the details of how to implement Step 4.2 on a CGM. Note that a continuous set E of nodes in a PQ- tree T is called *segregated* if $E = \text{leaves}(\text{lca}_r(E))$.

```
**Algorithm 4** Details of Step 4.2 in Algorithm 2.
**Input:** A set of $k+1$ $PQ$-trees $T_0, T_1, \cdots T_k$ with ground sets $S_0, S_1 \cdots S_k$, respectively, where $S_1 \cdots S_k$ are disjoint. Assume for $j = 1 \cdots k$ that $E_j = S_0 \cap S_j$ is non-empty and contiguous in both $T_0$ and $T_j$, and that $D_j = S_j - E_j$ is also non-empty and contiguous in $T_j$.
**Output:** A $PQ$-tree representing the join of $T_0$ with $T_1, \cdots T_k$.
    (1) For each $PQ$-trees $T_1, \cdots T_k$, segregate $T_0$ and $T_j$ with respect to $E_j$ [2]. If $|E_j| = 1$, let $\hat{T}_j = T_j$ and skip to Step 4.
    (2) For each $PQ$-tree $T_1, \cdots T_k$, find two elements $a_j$ and $b_j$ of $E_j$ that lie in different blocks of $T_0$'s partition of $E_j$ and different blocks of $T_j$'s partition of $E_j$. This can be done by applying the prefix sum algorithm. We assume here that $a_j$ precedes $b_j$ in the frontier, otherwise swap $a_j$ and $b_j$.
    (3) If $b_j$ precedes $a_j$ in the frontier of $T_j$ then let $\hat{T}_j = T_j$. Otherwise, $\hat{T}_j$ is computed from $T_j$ by flipping every node in $T_j$.
    (4) Segregate $D_j$ in $\hat{T}_j$. Replace the subtree $T_0|E_j$ of $T_0$ by $\hat{T}_j|D_j$, letting $z$ be the root of $\hat{T}_j|D_j$ in the resulting tree. If $E_j$ is rigid in both $T_0$ and $T_j$, then rename $z$ to be an $R$-node. If $E_j$ is hinged in $T_0$ but $z$ is an $R$-node, rename it to be a $Q$-node.
    (5) Output the resulting $PQ$-tree.
— End of Algorithm —
```

The CGM algorithm described above tests planarity of a graph G and computes a planar embedding, if possible, using $O(\log^2 p)$ communication rounds and linear sequential work per round.

Many important graph problems have been solved on the CGM model in $O(\log p)$ communication rounds [3]. To our knowledge there are no known BSP-like algorithms for the important graph planarity testing problem. This paper proposes a coarse-grained multicomputer algorithm for this problem, in $O(\log^2 p)$ communication rounds. It remains to be seen if this result can be improved, say, to $O(\log p)$ communication rounds. Our result implies a BSP algorithm with $O(\log^2 p)$ supersteps, $O(g \log^2(p) \frac{N}{p})$ communication, and $O(\log^2(p) \frac{N}{p})$ local computation.

References

[1] K.S. Booth and G.S. Lueker. Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using *PQ*-Tree Algorithms. *Journal of Computer and Systems Sciences*, 13:335-379, 1976.

[2] E. Cáceres, A. Chan, F. Dehne, G. Prencipe. Coarse grained parallel algorithms for detecting convex bipartite graphs. Technical Report, School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6, 1999.

[3] E. Cáceres, F. Dehne, A. Ferreira, P. Flocchini, I. Rieping, A. Roncato, N. Santoro, and S.W. Song. Efficient Parallel Graph Algorithms For Coarse Grained Multicomputers and BSP. Proceedings ICALP '97 - 24th International Colloquium on Automata, Languages, and Programming, P. Degano, R. Gorrieri, A. Marchetti-Spaccamela (editors), Bologna, Italy. *Lecture Notes in Computer Science*, 1256:390 - 400, Springer-Verlag. July 1997.

[4] F. Dehne (Ed.). Coarse grained parallel algorithms. *Algorithmica*, 24(3/4):173-426, 1999.

[5] F. Dehne and A. Chan. A note on coarse grained parallel integer sorting. *Proc. 13th Annual International Symposium on High Performance Computers (HPCS'99)*, Kingston, Canada, pp. 261-267, 1999.

[6] M.T. Goodrich. Communication efficient parallel sorting. *ACM Symposium on Theory of Computing* (STOC), 1996.

[7] J. Hopcroft and R. Tarjan. An efficient algorithm for graph planarity. *J. Assoc. Comput. Mach.*, 21:549-568, 1974.

[8] J. JáJá and J. Simon. Parallel algorithms in graph theory: Planarity testing. *SIAM J. Comput.* 11(2):313-328, 1982.

[9] P.N. Klein. An Efficient Parallel Algorithm for Planarity. Master's thesis, MIT, 1986.

[10] P.N. Klein. *Efficient Parallel Algorithms for Planar, Chordal, and Interval Graphs.* Ph.D. thesis, MIT, 1988.

[11] P.N. Klein, J.H. Reif. An Efficient Parallel Algorithm for Planarity. *Journal of Computer and System Sciences*, 37:190-246, 1988.

[12] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. in "Theory of Graphs: International Symposium: Rome, July, 1966" (P. Rosentiehl, Ed.), pp. 215-232, Gordon & Breach, New York, 1967.

[13] Y. Maon, B. Schieber, and U. Vishkin. Parallel Ear Decomposition Search (EDS) and *st*-Numbering in Graphs. *Theoretical Computer Science*, 47:277-298, 1986.

[14] T. Nishizeki. *Planar Graphs: Theory and Algorithms.* North Holland, 1988.

[15] V. Ramachandran and J.H. Reif. An optimal parallel algorithm for planarity. *Journal of Algorithms*, 1992.

[16] L. Valiant. A bridging model for parallel computation. *Communication of the ACM*, 33(8), August, 1990.