Purdue University
Purdue e-Pubs

Computer Science Technical Reports | Department of Computer Science

1987

Parallel Symmetry-Breaking in Sparse Graphs

Andrew V. Goldberg

Serge A. Plotkin

Gregory E. Shannon

Report Number:
87-710

Goldberg, Andrew V.; Plotkin, Serge A.; and Shannon, Gregory E., "Parallel Symmetry-Breaking in Sparse Graphs" (1987). *Computer Science Technical Reports.* Paper 614.
http://docs.lib.purdue.edu/cstech/614

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

TR 710

Parallel Symmetry-Breaking in Sparse Graphs

Andrew V. Goldberg*
Serge A Plotkin**
Gregory E. Shannon†

†Department of Computer Science
*Stanford University
**M.I.T.

CC-87-38 (December 1987)

Purdue University
West Lafayette, Indiana 47907

This research was supported in part by the Office of Naval Research under University Research Initiative grant number N00014-86-K-0689.

Parallel Symmetry-Breaking in Sparse Graphs*

Andrew V. Goldberg†
Dept. of Computer Science
Stanford University
Stanford, CA 94305

Serge A. Plotkin‡
Lab. for Computer Science
M.I.T.
Cambridge, MA 02139

Gregory E. Shannon§
Computer Sciences Dept.
Purdue University
West Lafayette, IN 47907

April 4, 1988

Abstract

We describe efficient deterministic techniques for breaking symmetry in parallel. These techniques work well on rooted trees and graphs of constant degree or genus. Our primary technique allows us to 3-color a rooted tree in $O(\lg^* n)$ time on an EREW PRAM using a linear number of processors. We use these techniques to construct fast linear processor algorithms for several problems, including the problem of $(\Delta + 1)$-coloring constant-degree graphs and 5-coloring planar graphs. We also prove lower bounds for 2-coloring directed lists and for finding maximal independent sets in arbitrary graphs.

1 Introduction

Some problems for which trivial sequential algorithms exist appear to be much harder to solve in a parallel framework. When converting a sequential algorithm to a parallel

*A preliminary version of this paper appeared in the Proc. of the 19th ACM Symp. on Theory of Computing, May 1987.

†Part of this research was done while the author was with at Lab. for Computer Science, MIT, Cambridge. Supported in part by a Fannie and John Hertz Foundation Fellowship and DARPA Contract N00014-80-C-0622.

‡Supported by DARPA Contract N00014-80-C-0622. Part of the work was done while the author was at AT&T Bell Laboratories, Murray Hill, NJ 07974.

§Supported in part by Hewlett-Packard's Faculty Development Program, NSF Grant DCR-8320124, and ONR Contract N00014-86-K-0689.

one, at each step of the parallel algorithm we have to choose a set of operations which may be executed in parallel. Often, we have to choose these operations from a large set of symmetrical operations, where interdependencies prevent simultaneous execution of all the operations in the set. Symmetry-breaking techniques enable the algorithm to select a large subset of independent operations.

Finding a maximal independent set (MIS) of a graph is a good example of the necessity of symmetry-breaking. At any step, a parallel MIS algorithm might have many candidate nodes to add to the independent set. However, due to adjacency constraints, not all of these nodes can be added simultaneously. A symmetry-breaking technique is therefore needed to find a large set of nodes to add, as has been done in previous parallel MIS algorithms [GS87,KW85,Lub86].

Previous symmetry-breaking techniques have focused on randomization. It is often desirable, however, to have a deterministic algorithm. Karp and Wigderson [KW85], and Luby [Lub86] proposed methods to convert certain randomized algorithms into deterministic ones. Their methods, however, significantly increase the number of processors used.

In many cases it is sufficient to break symmetry in sparse graphs. In this paper we introduce deterministic symmetry-breaking techniques for sparse graphs that use a linear number of processors. Our primary technique allows us to 3-color a rooted tree in $O(\lg^* n)$ time on a CREW PRAM. This technique was motivated by the deterministic coin-flipping technique developed by Cole and Vishkin [CV86].

We use our techniques to develop the linear-processor algorithms listed below.

- For graphs whose maximum degree is Δ, we give an $O((\lg \Delta)(\Delta^2+\lg^* n))$-time EREW PRAM algorithm for $(\Delta+1)$-coloring and for finding a maximal independent set.
- For planar graphs, we give 7-coloring, MIS, and maximal matching algorithms that run in $O(\lg n)$ time on a CRCW PRAM and in $O(\lg^2 n)$ time on an EREW PRAM.
- We give an $O(\lg n \lg^* n)$-time CRCW PRAM algorithm for 5-coloring an embedded planar graph.

The above results improve the running time and processor bounds for the respective

problems. The best deterministic linear-processor algorithm for finding MIS [GS87] runs in $O(\lg^4 n)$ time on constant-degree graphs, compared to $O(\lg^* n)$ time of our algorithm. The 5-coloring algorithms for planar graphs described in [BK87,Nao86] use $O(\lg^3 n)$ time and the same (large) number of processors as needed by Luby's MIS subroutine [Lub86]. The $O(\lg^3 n)$ running time of the maximal matching algorithm due to Israeli and Shiloach [IS86] can be reduced to $O(\lg^2 n)$ in the restricted case of planar graphs, but our algorithm is faster.

Although in this paper we have limited ourselves to the application of our techniques for the design of parallel algorithms for the PRAM model of computation, the same techniques can be applied in a distributed model of computation [Awe85,GHS83]. Moreover, the $\Omega(\lg^* n)$ lower bound for the MIS problem on a chain in the distributed model implies that our symmetry-breaking technique is optimal in this model [Awe87,Lin87].

Since we can 3-color a rooted tree in $O(\lg^* n)$ time, it is natural to ask if a rooted tree can be 2-colored as quickly. We answer this question by giving an $\Omega(\lg n/\lg\lg n)$ lower bound for 2-coloring of a rooted tree. We also present an $\Omega(\lg n/\lg\lg n)$ lower bound for finding a maximal independent set in a general graph, thus answering the question posed by Luby [Lub86].

This paper is organized as follows. In Section 2 we present definitions, notation, and computation model details. In Section 3 we present the algorithm for 3-coloring rooted trees. In Section 4 we use this algorithm to $(\Delta + 1)$-color constant-degree graphs. In Section 5 we use results of Section 4 to develop algorithms for planar graphs. In Section 6 we prove the lower bounds mentioned earlier.

2 Preliminaries

This section describes the assumptions about the computational model and introduces the notation used throughout the paper. We consider simple, undirected graphs with n vertices and m edges. The maximum degree of a graph is denoted by Δ. The graph induced by a set of nodes X is denoted by $G[X]$.

Given a graph $G = (V, E)$, we say that a subset of nodes $I \subseteq V$ is *independent* if no two nodes in I are adjacent. A *coloring* of a graph G is an assignment $C : V \to I^+ \cup \{0\}$ of nonnegative (not necessarily consecutive) integers (colors) to nodes of the graph. A coloring is *valid* if no two adjacent nodes have the same color. The bits are numbered from 0, and the i^{th} bit in the color of a node v is denoted by $C_v(i)$. A subset of edges $M \subseteq E$ is a matching if each pair of distinct edges in M have no nodes in common.

The following problems are discussed in the paper:

- The node-coloring problem: find a valid coloring of a given graph that uses at most $\Delta+1$ colors.
- The maximal independent set (MIS) problem: find a maximal independent set of vertices in a given graph.
- The maximal matching (MM) problem: find a maximal matching in a given graph.

We make a distinction between *unrooted* and *rooted* trees. In a rooted tree, each nonroot node knows which of its neighbors is its parent.

The following notation is used:

$$\begin{aligned} \lg x &= \log_2 x \\ \lg^{(1)} x &= \lg x \\ \lg^{(i)} x &= \lg\lg^{(i-1)} x \\ \lg^* x &= \min\{i \mid \lg^{(i)} x \leq 2\} \end{aligned}$$

We assume a PRAM model of computation [BH85,FW78] where each processor is capable of executing simple word and bit operations. The word width is assumed to be $O(\lg n)$. The word operations we use include bit-wise boolean operations, integer comparisons, and unary-to-binary conversion. Each processor P has a unique *identification number* $O(\lg n)$ bits wide, which we denote by PE-ID(P). We use adjacency lists to represent the graph, assigning a processor to each edge and each node of the graph. We use exclusive-read exclusive-write (EREW) PRAM, concurrent-read exclusive-write (CREW) PRAM, and concurrent-read concurrent-write (CRCW) PRAM, as appropriate. The write conflicts in CRCW PRAM are assumed to be resolved arbitrarily. All lower bounds are proven for a CRCW PRAM with a polynomial number of processors.

```
Procedure 6-Color-Rooted-Tree(T)
    N_c ← n;
    for all v ∈ V in parallel do C_v ← PE-ID(v);
    while N_c > 6 do
          for all v ∈ V in parallel do begin
              if v is the root then begin
                i_v ← 0;
                b_v ← C_v(0);
              end;
              else begin
                i_v ← min{i | C_v(i) ≠ C_parent(v)(i)};
                b_v ← C_v(i_v);
              end;
              C_v ← i_v b_v;
          end;
          N_c ← max{C_v | v ∈ V} + 1;
    end;
end.
```

Figure 1: The Coloring Algorithm for Rooted Trees

3 Coloring Rooted Trees

This section describes an $O(\lg^* n)$-time algorithm for 3-coloring rooted trees. We first describe an $O(\lg^* n)$-time algorithm for 6-coloring rooted trees and then show how to transform a 6-coloring of a rooted tree into a 3-coloring in constant time.

The procedure *6-Color-Rooted-Tree* is shown in Figure 1. This procedure accepts a rooted tree $T = (V, E)$ and 6-colors it in time $O(\lg^* n)$. Starting from the valid coloring given by the processor ID's, the procedure iteratively reduces the number of bits in the color descriptions by recoloring each nonroot node v with the color obtained by concatenating the index of a bit in which C_v differs from $C_{parent(v)}$ and the value of this bit. The root r concatenates 0 and $C_r[0]$ to form its new color.

Theorem 1 *The algorithm* 6-Color-Rooted-Tree *produces a valid 6-coloring of a tree in* $O(\lg^* n)$ *time on a CREW PRAM using a linear number of processors.*

Proof: First we prove by induction that the coloring computed by the algorithm is valid, and then we prove the upper bound on the execution time.

Assuming that the coloring C is valid at the beginning of an iteration, show that the coloring at the end of the iteration is also valid. Let v and w be two adjacent nodes with v being the parent of w. In the algorithm, w chooses some index i such that $C_v(i) \neq C_w(i)$ and v chooses some index j such that $C_v(j) \neq C_{parent(v)}(j)$. The new color of w is $\langle i, C_w(i) \rangle$ and the new color of v is $\langle j, C_v(j) \rangle$. If $i \neq j$, the new colors are different and we are done. On the other hand, if $i = j$, then $C_v(i)$ can not be equal to $C_w(i)$ by the definition of i, and again the colors are different. Hence, the validity of the coloring is preserved.

Now we show that the algorithm terminates after $O(\lg^* n)$ iterations. Let L_k denote the number of bits in the representation of colors after k iterations. For $k = 1$ we have

$$\begin{aligned} L_1 &= \lceil \lg L \rceil + 1 \\ &\leq 2\lceil \lg L \rceil \end{aligned}$$

if $\lceil \lg L \rceil \geq 1$.

Assume for some k we have $L_{k-1} \leq 2\lceil \lg^{(k-1)} L \rceil$ and $\lceil \lg^{(k)} L \rceil \geq 2$. Then

$$\begin{aligned} L_k &= \lceil \lg L_{k-1} \rceil + 1 \\ &\leq \lceil \lg(2 \lg^{(k-1)} L) \rceil + 1 \\ &\leq 2\lceil \lg^{(k)} L \rceil \end{aligned}$$

Therefore, as long as $\lceil \lg^{(k)} L \rceil \geq 2$,

$$L_k \leq 2\lceil \lg^{(k)} L \rceil.$$

Hence, the number of bits in the representation of colors L_k decreases until, after $O(\lg^* n)$ iterations, $\lceil \lg^{(k)} L \rceil$ becomes 1 and L_k reaches the value of 3 (the solution of $L = \lceil \lg L \rceil + 1$). Another iteration of the algorithm produces a 6-coloring: 3 possible values of the index i_v and 2 possible values of the bit b_v. The algorithm terminates at this point.

Using concurrent-read, each node determines its parent's color in constant time. Given two colors, C_v and C_w, we can compute the smallest index j such that the j-th bit of C_v differs from the j-th bit of C_w by computing $j =$ unary-to-binary($|C_v - C_w|$ XOR ($|C_v -$

$C_w| - 1)$). Hence, each node can compute the new color independently in constant time. Therefore, each iteration takes constant time and the algorithm uses $O(\lg^* n)$ time overall. Note that no concurrent-write capabilities are required; for constant-degree trees the concurrent-read capability is not needed either. ∎

We now describe the algorithm *3-Color-Rooted-Tree* which 3-colors a rooted tree. The algorithm first applies *6-Color-Rooted-Tree* to produce a valid 6-coloring of the tree. Then it executes three stages, each time reducing the number of colors by one.

Each stage works as follows. By *shifting down* the coloring we mean recoloring each nonroot node with the color of its parent and recoloring the root with a color different from its current color. To remove the color $c \in \{3, 4, 5\}$, first shift down the current coloring. Then, recolor each node of color c with the smallest color different from its parent's and children's colors.

Theorem 2 *Given a rooted tree T, the algorithm* 3-Color-Rooted-Tree *constructs a valid 3-coloring of T using n processors and $O(\lg^* n)$ time on a CREW PRAM.*

Proof: After a shift of colors, the children of any node have the same color. Thus each node is adjacent to nodes of at most two different colors. Therefore, each stage of the algorithm reduces the number of colors by one, as long as the number of colors is greater than three. Each stage takes a constant time on a CREW PRAM. The theorem follows from Theorem 1. ∎

To describe the subsequent algorithms, we introduce the concept of a pseudoforest [PQ82]. A *pseudoforest* of $G = (V, E)$ is a directed graph $G' = (V, E')$, such that $(u, v) \in E' \Rightarrow \{u, v\} \in E$ and outdegree of any node is at most one. A *maximal pseudoforest* of $G = (V, E)$ is a directed graph $G' = (V, E')$, such that $(u, v) \in E' \Rightarrow \{u, v\} \in E$ and outdegree of any node in G' is one, unless this node is zero-degree in G. Nodes with zero out-degree are *roots* of the pseudoforest. We assume that graphs are represented by adjacency lists, and therefore a maximal pseudoforest can be constructed in (parallel) constant time by choosing an arbitrary adjacent edge for every node and directing this edge outward.

The coloring algorithms presented in this section work for pseudoforests as well as for rooted trees. Therefore, a pseudoforest can be 3-colored in $O(\lg^* n)$ time on an CRCW PRAM using a linear number of processors. We shall call the procedure for 3-coloring pseudoforests *3-Color-Pseudoforest*. Note that an odd cycle is a pseudoforest that can not be colored in less than 3 colors, and therefore the number of colors used by the procedure *3-Color-Pseudoforest* is optimal in this case.

Any tree can be 2-colored. In fact, it is easy to 2-color a tree in polylogarithmic time. For example, one can use treefix operations [LM86,MR85] to compute the distance from each node to the root, and color even level nodes with one color and odd level nodes with the other color. It is harder to find a 2-coloring of a rooted tree in parallel, however, than it is to find a 3-coloring of a rooted tree. In section 6 we show a lower bound of $\Omega(\lg n / \lg\lg n)$ on 2-coloring of a directed list on a CRCW PRAM with a polynomial number of processors, which implies the same lower bound for rooted trees.

4 Coloring Constant-Degree Graphs

The method for coloring rooted trees, described in the previous section, is a generalization of the deterministic coin-flipping technique described in [CV86]. The method can be generalized even further [GP87b] to color constant-degree graphs in a constant number of colors. In the generalized algorithm, a current color of a node is replaced by a new color obtained by looking at each neighbor, appending the index of a bit in which the current color of the node is different from the neighbors' color to the value of the bit in the node color, and concatenating the resulting strings. This algorithm runs in $O(\lg^* n)$ time, but the number of colors, although constant as a function of n, is exponential in the degree of the graph.

In this section we show how to use the procedure *3-Color-Pseudoforest*, described in the previous section, to color a constant-degree graph with $(\Delta+1)$ colors.

The algorithm *Color-Constant-Degree-Graph* which colors a constant-degree graph $G = (V, E)$ with $(\Delta+1)$ colors is presented in Figure 2. The algorithm consists of two phases. In the first phase we iteratively construct a maximal pseudoforest and remove its edges

```
Procedure Color-Constant-Degree-Graph.
    E' ← {(v,w) | {v,w} ∈ E};
    for i = 0 to Δ do begin                                   ⟨⟨ the first phase⟩⟩
        for all v ∈ V in parallel do
            if ∃(v,u) ∈ E' then E_i ← E_i + (v,u) ;
        E' ← E' − E_i;                                        ⟨⟨ E_i are edges of a maximal pseudoforest ⟩⟩
    end;
    for all v ∈ V in parallel do                              ⟨⟨ initial coloring⟩⟩
        C(v) ← 0;
    for all 0 ≤ i ≤ Δ in parallel do                          ⟨⟨ color the pseudoforests⟩⟩
        C_i ← 3-Color-Pseudoforest(V, E_i);
    for i ← Δ down to 0 do begin                              ⟨⟨ the second phase⟩⟩
        E' ← E' + E_i;
        for k ← 1 to 2, j ← 0 to Δ do
            V' ← V;
            for all v ∈ V' in parallel do
                if C(v) = j and C_i(v) = k
                  then begin
                  C(v) ← max{{0,1,...,Δ} − {C(w) | (v,w) ∈ E'}};
                  V' ← V' − {v};
                end;
            end;
        end;
    end;
end.
```

Figure 2: The Coloring Algorithm for Constant Degree Graphs

from G. This phase continues until no edges remain, at which point we color all the nodes with one color. Then we color all the pseudoforests with 3 colors in parallel.

In the second phase we iteratively return the edges of the current pseudoforest, each time recoloring the nodes to maintain a consistent coloring. At the beginning of each iteration of this phase, the edges E' of the current pseudoforest are added, making the existing $(\Delta+1)$-coloring inconsistent. The forest E' is already colored with 3 colors. Now, each node has two colors – one from the coloring at the previous iteration and one from the coloring of the forest. The pairs of colors form a valid $3(\Delta+1)$-coloring of the graph. The iteration finishes by enumerating the color classes, recoloring each node of the current color with a color from $\{0,\ldots,\Delta\}$ that is different from the colors of its neighbors. We can recolor all the nodes of the same color in parallel because they are independent.

Theorem 3 *The algorithm* Color-Constant-Degree-Graph *colors the graph with* $(\Delta+1)$ *colors and runs in* $O((\lg\Delta)(\Delta^2+\lg^* n))$ *time on an EREW PRAM using a linear number of processors.*

Proof: At each iteration all edges of the maximal pseudoforest are removed. The definition of a maximal pseudoforest implies that each node that still has neighbors in the beginning of an iteration has at least one edge removed during that iteration and therefore its degree decreases. After at most Δ iterations, E' is empty. The running time of each iteration is determined by the time required to select an unused edge out of an edge list. On an EREW PRAM, an unused edge can be selected in $O(\lg\Delta)$ time. The pseudoforests are edge-disjoint and therefore can be colored in parallel. By Theorem 2, this takes $O(\lg\Delta\lg^* n)$ time on an EREW PRAM. The $\lg\Delta$ factor appears because we do not use the concurrent-read capability; a node must broadcast its color to its children using, for example, recursive doubling. The total time bound for the first stage is therefore $O((\lg\Delta)(\Delta+\lg^* n))$.

The second phase terminates in at most Δ iterations as well. For each pseudoforest we iterate over all the colors. Since in this section we assume that Δ is a constant, each iteration can be done in $O(\lg\Delta)$ time using word operations; for example, we can represent colors as bit vectors and use exclusive-or function together with the recursive doubling technique. Hence, one iteration of the second phase takes $O(\Delta\lg\Delta)$ time, leading

to an overall $O((\lg\Delta)(\Delta^2+\lg^* n))$ running time for the second stage of the algorithm and for the algorithm itself. ∎

Given a $(\Delta+1)$-coloring of a graph, we can find an MIS of the graph by iterating over the colors, taking all the remaining nodes of the current color, adding them to the independent set, and removing them and all their neighbors from the graph. (We refer to this procedure as *Constant-Degree-MIS* in the subsequent sections.) The running time of this algorithm is dominated by the running time of the *Color-Constant-Degree-Graph* procedure. The following theorem states this fact formally.

Theorem 4 *An MIS in constant-degree Δ graphs can be found in $O((\lg\Delta)(\Delta^2+\lg^* n))$ time on an EREW PRAM using a linear number of processors.*

Remark: The proofs of Theorems 3 and 4 imply that the algorithms *Color-Constant-Degree-Graph* and *Constant-Degree-MIS* have a polylogarithmic running times for graphs with polylogarithmic maximum degrees. For graphs with arbitrary maximum degree we can use the following algorithm. First, the graph is partitioned into two subgraphs with approximately equal number of nodes, and the subgraphs are recursively colored in $\Delta+1$ colors. Then we iterate through all the colors of one of the subgraphs, recoloring each node with a color different from the colors of all of its neighbors. We can find this color using sorting in $O(\lg\Delta)$ time [Col86]. This algorithm colors a graph with a maximum degree of Δ with $\Delta+1$ colors in $O(\Delta\lg\Delta\lg n)$ time.

The above algorithms can be implemented in the distributed model of computation [Awe85,GHS83], where processors have fixed connections determined by the input graph. The algorithms in the distributed model achieve the same $O(\lg^* n)$ bound as in the EREW PRAM model. It was recently shown that $\Omega(\lg^* n)$ time is required in the distributed model to find a maximal independent set on a chain [Awe87,Lin87]. Our algorithms are therefore optimal (to within a constant factor) in the distributed model.

In [Sha86], *flat forests* are used to develop a linear processor constant-degree MIS algorithm which used time exponential in Δ. A forest is *flat* if each of its trees, when properly oriented, has a height of at most 1, and any zero-degree node in the forest is zero-

degree in the input graph. Using the techniques introduced in this section, we can find a flat forest of a graph by proceeding as follows. Find a maximal pseudoforest $P = G(V, E')$. Note that there exists a flat forest $F = (V, E'')$, such that $E'' \subseteq E'$. Use the algorithm *3-Color-Pseudoforest* to find a 3-coloring of the pseudoforest P and subsequently find an MIS I of P. Each node $v \notin I$ adds an edge (v, u) to E'' such that $u \in I$. Each node in I with no adjacent edges in E'', but some adjacent edges in E', chooses one adjacent edge in E' and adds it to E''. The graph F induced by the edges in E'' is almost a flat forest – each tree has a height of at most 2. Now we split trees of height 2 in F into trees of height one to produce a flat forest. All operations take constant time except the operation of finding the 3-coloring of P, which takes $O(\lg^* n)$ time. Therefore, we can find a flat forest in $O(\lg^* n)$ time on a CREW PRAM using n processors.

5 Coloring and Matching in Planar Graphs

Euler's formula [Har72] implies that every planar graph has a constant fraction of nodes of degree 6 or less. In this section we use this property in conjunction with the techniques developed above to construct efficient algorithms for coloring and finding maximal matchings in planar graphs.

First we present the algorithm *7-Color-Planar-Graph* which finds a 7-coloring of a planar graph in $O(\lg n)$ time. The algorithm is shown in Figure 3. The first stage of the algorithm partitions the nodes of the graph into sets V_i, such that the degree of any node $v \in V_i$ in $G[V_i + V_{i+1} + V_{i+2}, \ldots]$ is at most 6 (V_i consists of all nodes of degree at most 6 in $G[V - (V_0 \cup V_1 \cup \ldots \cup V_{i-1})]$). Then, the algorithm colors all the subgraphs induced by the node-sets $\{V_i\}$. These graphs are node-disjoint and therefore the coloring can be done in parallel. The last stage of the algorithm adds the subgraphs back in reverse order, updating the coloring.

Theorem 5 *The algorithm* 7-Color-Planar-Graph *constructs a valid 7-coloring using n processors and $O(\lg n)$ time on a CRCW PRAM.*

Proof: By Euler's formula, at least a constant fraction of any planar graph's nodes are of

```
Procedure 7-Color-Planar-Graph
    V' ← V;
    V_1, V_2, ... V_{5⌈lg n⌉} ← ∅;
    i ← 0;
    while V' ≠ ∅ for all v ∈ V' do in parallel                    ⟨⟨ first stage ⟩⟩
        if Degree(v) ≤ 6
          then begin
                 V_i ← V_i + v;
                 V' ← V' − v;
          end;
        i ← i + 1;
    end;
    k ← i − 1;
    for all 0 ≤ i ≤ k do in parallel                              ⟨⟨ color the pseudoforests⟩⟩
        E_i ← {{v, w} | v, w ∈ V_i ; {v, w} ∈ E};
        C_i ← Color-Constant-Degree-Graph(V_i, E_i);
    end;
    for i ← k down to 0 do                                        ⟨⟨ second stage ⟩⟩
        V'' ← V_i;
        for j ← 0 to 6 do
            for all v ∈ V'' do in parallel
                if C_v = j
                  then begin
                         C_v ← max{{0, ..., 6} − {C_w | w ∈ V'; {v, w} ∈ E} };
                         V'' ← V'' − v;
                         V' ← V' + v;
                  end;
                end;
            end;
        end;
    end.
```

Figure 3: The 7-Coloring Algorithm For Planar Graphs

degree 6 or less. Therefore, the first stage partitions the nodes of G into at most $O(\lg n)$ sets V_i. We use concurrent reads and writes to determine whether the degree of a node is at most 6, and hence each iteration of the first stage is done in constant time. By Theorem 3, the second stage uses only $O(\lg^* n)$ time. In the i^{th} iteration of the third stage, the graph $G[V_i]$ is already 7-colored and the maximum degree of each node in V_i in the graph $G[V_i + V_{i+1} + V_{i+2} + \ldots]$ is at most 6. Only constant time is then needed to add in V_i and produce a valid 7-coloring of $G[V_i + V_{i+1} + V_{i+2} + \ldots]$. Therefore, only $O(\lg n)$ time is used in all three stages. ∎

Remark: If at the first stage, instead of removing from the graph all the nodes with degree of at most 6, we remove all nodes with degree of at most c times average degree (for $c > 1$), the algorithm described above runs in polylogarithmic time for any graph G such that the average degree of any node-induced subgraph G' of G is polylogarithmic in the size of G'. This class contains many important subclasses including graphs that are unions of a polylogarithmic number of planar graphs (i.e. graphs with polylogarithmic thickness [Har72]).

Given a valid 7-coloring of a planar graph, we can find an MIS in the graph by iterating through colors as in our *Constant-Degree-MIS* algorithm. With concurrent reads and writes, only constant time is needed for each color class. Hence, we can find an MIS in a planar graph in $O(\lg n)$ time on a CRCW PRAM using a linear number of processors.

The deterministic parallel algorithms for 5-coloring planar graphs, described in [BK87] and in [Nao86], use $\Omega(\lg^3 n)$ time and $O(n^3)$ processors. These algorithms require a large number of processors because they use Luby's MIS algorithm [Lub86]. Using the *Constant-Degree-MIS* algorithm described in the previous section, we can reduce the number of processors to linear, while increasing the running time by a factor of $O(\lg^* n)$ [GP87a, Gol87].

The 5-coloring algorithm presented below is essentially a parallelization of the sequential algorithms in [CNS81,MST80]. Given an embedding (which can be computed in $O(\lg^2 n)$ time [KR86]), our algorithm runs in $O(\lg n \lg^* n)$ time on a CRCW PRAM using a linear number of processors. Given a graph $G = (V, E)$, the algorithm finds a special

large independent set I of nodes in G, merges some of the neighbors of I (as described below) and removes the nodes in I to create a new graph G', recursively colors G', and uses this coloring to color the nodes in G.

The special independent set I is constructed as follows. Let Q be the set of all nodes in G of degree greater than 42. Let V_4 be the set of all nodes of degree 4 or less. Let V_5 and V_6 be the set of all nodes of degree 5 with at most one neighbor in Q and the set of all nodes of degree 6 with no neighbors in Q, respectively. Let $S = V_4 \cup V_5 \cup V_6$. Let $G^s = (S, E^s)$ be the graph induced by the nodes in S in the graph which is the square of $G[V - Q]$ The set I is a maximal independent set in the graph $G^s \cup G[V - Q]$. Since G^s and $G[V - Q]$ are of constant degree, we can find I using the procedure *Constant-Degree-MIS*.

In order to construct the graph G', the algorithm proceeds as follows. Start with $G' = G$. For each node in $I \cap V_5$ we find two of its non-adjacent neighbors that have low degree (42 or less), and merge them into a single supernode. For each node in $I \cap V_6$ we either merge three of its non-adjacent neighbors into a single supernode, or merge two non-adjacent pairs of its neighbors into two supernodes. The embedding information is used as in [CNS81,MST80] to find the neighbors that can be merged while preserving planarity after all nodes in I are removed. Then we remove all the nodes in I to get the graph G'.

After recursively 5-coloring the graph G', we obtain the coloring of G as follows. First we color all the nodes of G that correspond to nodes or supernodes of G' with the same color they were colored in G'. Now we add all the nodes in I and in parallel color every one of them with a color different from the colors of its neighbors.

In order to bound the running time of the 5-coloring algorithm we need the following lemma, which is similar to Lemma 3 in [CNS81].

Lemma 6 *The size of $S = V_4 \cup V_5 \cup V_6$ is at least a constant fraction of the total number of nodes in the graph.*

Proof: Let $R = V - S$. Denote by s_i and r_i the number of nodes of degree i in the sets S and R, respectively. Let $r_* = \sum_{i=7}^{42} r_i$, and let $r_Q = \sum_{i=43}^{\infty} r_i$. By Euler's formula, $r_Q \leq \frac{6}{43}n$.

We prove the lemma by a counting argument. Definitions of r_5 and r_6 imply that

$2r_5 + r_6 \leq \sum_{i=43}^{\infty} ir_i$. Euler's formula implies that $3n \geq m$, therefore

$$\begin{aligned}
6n &\geq \sum_{i=1}^{6} is_i + 5r_5 + 6r_6 + \sum_{i=7}^{42} ir_i + \sum_{i=43}^{\infty} ir_i \\
&\geq \sum_{i=1}^{6} is_i + 7r_5 + 7r_6 + \sum_{i=7}^{42} ir_i \\
&\geq 7r_5 + 7r_6 + 7\sum_{i=7}^{42} r_i \\
&\geq 7(\sum_{i=1}^{6} s_i + r_5 + r_6 + r_* + r_Q) - 7\sum_{i=1}^{6} s_i - 7r_Q \\
&\geq 7n - 7|S| - 7 \cdot \frac{6}{43}n
\end{aligned}$$

Thus $|S| \geq \frac{n}{301}$. ∎

Theorem 7 *Given an embedded planar graph, the algorithm* 5-Color-Planar-Graph *5-colors it using n processors in $O(\lg^* n \lg n)$ time on a CRCW PRAM, and $O((\lg^* n + \lg \Delta) \lg n)$ time on an EREW PRAM.*

Proof: Correctness of the algorithm follows from [CNS81] and from the fact that the nodes in I are independent in $G^s \cup G[V - Q]$

Lemma 6 implies that the size of S is $\Omega(n)$. The graph G^s has a constant maximum degree and hence the size of the set I is $\Omega(n)$ as well. Therefore the depth of recursion is at most $O(\lg n)$.

On a CRCW PRAM, we can find S and Q in constant time as in the algorithm *7-Color-Planar-Graph*. The construction of $G^s \cup G$ takes constant time because G^s has constant degree. The algorithm *Constant-Degree-MIS* finds I in $O(\lg^* n)$ time. In constant time nodes in I can merge appropriate neighbors and delete themselves from G to form G'. Edge lists in G' need not be compacted when we are using the CRCW PRAM. After recursively coloring G', we can color G in constant time.

On the EREW PRAM, $O(\lg \Delta)$ additional time per recursion level is needed since we must compact edge lists of G' (so that the set S in G' can be found in constant time). ∎

Remark: Chrobak, Diks, and Hagerup [CDH87] have recently improved the result of Theorem 7 by giving an algorithm for 5-coloring planar graphs that runs in $O(\lg^* n \lg n)$ time on an EREW PRAM and does not need an embedding.

Using the techniques described in this section, it is easy to construct a fast algorithm for finding a maximal matching (MM) in a planar (or a constant-degree) graph. As in the 7-coloring algorithm, the first stage of the MM algorithm separates the nodes of the graph into sets V_i, such that the degree of any node $v \in V_i$ in $G[V_i + V_{i+1} + V_{i+2}, \ldots]$ is at most 6. Then the graphs $\{G[V_i]\}$ are colored in parallel. The second stage of the algorithm recursively finds MM in the graph $G[V - V_1]$ and removes the matched nodes to get $G[V']$, where V' is the set of the unmatched nodes. The graph $G[V']$ has no edges and the nodes V_1 in the graph $G[V' + V_1]$ have maximum degree of 6. Hence, in 7 iterations over the colors of $G[V_1]$ we can find the MM of G.

Theorem 8 *A maximal matching in a planar graph can be found in $O(\lg n)$ time on a CRCW PRAM using a linear number of processors.*

Remark: Using Euler's formula, we can extend our algorithms for 7-coloring and MIS in planar graphs to graphs of bounded genus γ. We apply the algorithm *7-Color-Planar-Graph* as before when there are at least $c\gamma$ nodes remaining in the residual graph, for some constant c. The Heawood map-coloring theorem states that any graph can be colored with $O(\sqrt{\gamma})$ colors, and its proof implies a polynomial time algorithm for finding such a coloring [Har72]. Therefore, when less than $c\gamma$ nodes remain in the residual graph, we sequentially color it with $O(\sqrt{\gamma})$ colors. With additional time that is polynomial in γ, we can then $O(\sqrt{\gamma})$-color the graph using the same time and number of processors as for 7-coloring a planar graph. Note, that the above algorithm does not need embedding. The related result for MIS on bounded-genus graphs follows as before.

6 Lower Bounds

In this section we prove two lower bounds for a CRCW PRAM with a polynomial number of processors:

- Finding a maximal independent set in a general graph takes $\Omega(\lg n/\lg\lg n)$ time.
- 2-coloring a directed list takes $\Omega(\lg n/\lg\lg n)$ time.

The first lower bound complements the $O(\lg n)$ CRCW PRAM upper bound for the MIS problem that is achieved by Luby's algorithm [Lub86]. The second lower bound complements Theorem 2 in this paper.

Theorem 9 *The running time of any MIS algorithm on a CRCW PRAM with a polynomial number of processors is* $\Omega(\lg n/\lg\lg n)$.

Proof: Given an instance of MAJORITY, we construct an instance of MIS in constant CRCW PRAM time. MAJORITY is harder that PARITY [FSS81], which was proven to take $\Omega(\lg n/\lg\lg n)$ time on a CRCW PRAM in [Bea86,BH87]. Therefore the lower bound claimed in the theorem follows.

Let $x_1, x_2, \ldots, x_n$ be an instance of MAJORITY. We construct a complete bipartite graph $G = (V, E)$ with nodes corresponding to '0' bits of the input on one side and nodes corresponding to '1' bits on the other side.

$$\begin{aligned} V &= \{1, \ldots, n\} \\ E &= \{(i,j) \mid x_i \neq x_j\} \end{aligned}$$

To construct this graph, assign a processor P_{ij} for each pair $1 \leq i < j \leq n$. Then, each processor P_{ij} writes 1 into location M_{ij} if $x_i \neq x_j$ and writes 0 otherwise.

A maximal matching in a complete bipartite graph is also a maximum one. By constructing a maximal independent set in the line-graph G' of G, one can find a maximal matching in G. To construct the graph G' assign a processor P_{ijk} for each distinct $i, j, k \leq n$. Each P_{ijk} writes 1 into location $M_{(i,j),(j,k)}$ if $M_{ij} = M_{jk} = 1$ and 0 otherwise.

The MAJORITY equals to 1 if and only if there is an unmatched node $i \in G$ such that $x_i = 1$, which can be checked on a CRCW PRAM in constant time. ∎

Theorem 10 *The time to 2-color a directed list on a CRCW PRAM with a polynomial number of processors is* $\Omega(\lg n / \lg\lg n)$.

Proof: We show a constant time reduction from PARITY to the 2-coloring of a directed list. First, we show how to construct, in constant time, a directed list with elements corresponding to all the input bits x_i with value of 1. Let $x_1, x_2, \ldots, x_n$ be an instance of PARITY. We can assume w.l.o.g. that $x_1 = 1$. With each input cell M_i (that initially holds the value of x_i), associate a processor P_i, a set of processors P_i^{jk} with each index i, $1 \leq k \leq j < i$, a set of cells M_i^j, $0 \leq j < i$, and a cell M_i'. Initialize all cells that do not store input bits to 0.

In one step, each processor P_i^{jk} reads the value of M_{i-k} and, if it equals to 1, writes 1 into M_i^j, effectively computing the OR-function on the input values $x_{i-j}, x_{i-j+1}, \ldots, x_{i-1}$. Assign a processor P_i^j to each M_i^j, $1 \leq j < i$. Each processor P_i^j reads M_i^j and M_i^{j+1} and writes $(i-j)$ into M_i' if and only if $M_i^j \neq M_i^{j-1}$. It can be seen that for all $1 \leq i \leq n$, M_i' holds $\max\{j \mid j < i, x_j = 1\}$.

We have constructed a directed list with elements corresponding to all the input bits x_i with value of 1. Assume this list is 2-colored. Then PARITY equals to 1 if and only if both ends of the list are colored with the same color, which can be checked in constant time. ∎

7 Conclusion and Open Problems

We have presented a fast technique for breaking symmetry in parallel and have shown how to apply this technique to improve the running times and processor bounds of a number of important parallel algorithms. We believe that the efficiency of this technique, combined with the simplicity of its implementation, makes it an important tool in designing parallel

algorithms.

Our results motivate the following open questions.

- We have proved a lower bound for MIS in general graphs. What is the lower bound for MIS in planar graphs ?
- Beame has proposed the following algorithm for coloring rooted trees of constant degree on PRAM. Run the algorithm *3-Color-Rooted-Tree* for $O(\lg\lg^* n)$ steps. Next, each processor collects the colors of all the descendants on distance $O(\lg^* n)$ or less and uses this information and a precomputed lookup table (of size $O(\lg^* n \lg\lg^* n)$) to compute its final color. Given an $\Omega(\lg^* n)$ preprocessing time, we can precompute the lookup table; after this preprocessing step, the time to 3-color a tree (or a pseudoforest) will be $O(\lg\lg^* n)$. Is it possible to 3-color a tree in $o(\lg^* n)$ time on PRAM with no preprocessing?
- Can we compute an MIS in general graphs in $o(\lg n)$ time ?
- Recently, several papers [CV86,Rei85] that present parallel algorithms with "optimal speedup" have been published. (The measure of optimality used in these papers is how close is the processor-time product of the parallel algorithm to the running time of the fastest known sequential one.) The problems we have been studying in this paper can be solved in linear sequential time, but the processor-time products achieved by our algorithms are superlinear (by a $\lg^* n$ or a polylogarithmic factor). How can one reduce the processor requirements of the algorithms without increasing their running time in order to achieve linear time-processor products? Also, will this improved processor efficiency induce significantly more constraints on the model as compared to our current algorithms ?

Acknowledgments

We would like to thank Greg Frederickson, Charles Leiserson, and David Shmoys for fruitful and stimulating discussions, and for their valuable comments on a draft of this paper. We are also grateful to the referee whose extremely thorough comments have significantly improved the presentation of the paper.

References

[Awe85] B. Awerbuch. Complexity of network synchronization. *Journal of the Association for Computing Machinery*, 32(4):804–823, October 1985.

[Awe87] B. Awerbuch. A tight lower bound on the time of distributed maximal independent set algorithms. February 1987. Unpublished manuscript.

[Bea86] P. Beame. *Lower Bounds in Parallel Machine Computation.* PhD thesis, University of Toronto, 1986.

[BH85] A. Borodin and J. E. Hopcroft. Routing, merging, and sorting on parallel models of computation. *Journal of Computer and System Sciences*, 30:130–145, 1985.

[BH87] P. Beame and J. Hastad. Optimal bounds for decision problems on the CRCW PRAM. In *Proc. 19th ACM Symp. on Theory of Computing*, pages 83–93, 1987.

[BK87] J. Boyar and H. Karloff. Coloring planar graphs in parallel. *J. of Algorithms*, 1987. (To appear).

[CDH87] M. Chrobak, K. Diks, and T. Hagerup. Parallel 5-coloring of planar graphs. In 14^{th} *International Colloquium on Automata, Languages, and Programming*, pages 304–313, July 1987.

[CNS81] N. Chiba, T. Nishizeki, and N. Saito. A linear 5-color algorithm of planar graphs. *Journal of Algorithms*, 2:317–327, 1981.

[Col86] R. Cole. Parallel merge sort. In *Proc. 27th IEEE Annual Symposium on Foundations of Computer Science*, pages 511–516, 1986.

[CV86] R. Cole and U. Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70:32–56, 1986.

[FSS81] M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial time hierarchy. In *Proc. 22nd IEEE Symp. on Foundations of Computer Science*, pages 260–270, 1981.

[FW78] S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proc. 10th ACM Symp. on Theory of Computing*, pages 114–118, 1978.

[GHS83] R. G. Gallager, P. A. Humblet, and P. M Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and Systems*, 5(1):66–77, January 1983.

[Gol87] A. V. Goldberg. *Efficient Graph Algorithms for Sequential and Parallel Computers.* PhD thesis, M.I.T., January 1987.

[GP87a] A. Goldberg and S. Plotkin. *Efficient Parallel Algorithms for* $(\Delta+1)$*-Coloring and Maximal Independent Set Problems.* Technical Report MIT/LCS/TM-320, MIT, January 1987.

[GP87b] A. Goldberg and S. Plotkin. Parallel $(\Delta+1)$ coloring of constant-degree graphs. *Information Processing Letters*, 25(4):241–245, June 1987.

[GS87] M. Goldberg and T. Spencer. A new parallel algorithm for the maximal independent set problem. In *Proc. 28th IEEE Symp. on Foundations of Computer Science*, 1987. (To appear).

[Har72] F. Harary. *Graph Theory.* Addison-Wesley, 1972.

[IS86] A. Israeli and Y. Shiloach. An improved parallel algorithm for maximal matching. *Information Processing Letters*, 22:57–60, January 1986.

[KR86] P. Klein and J. Reif. An efficient parallel algorithm for planarity. In *Proc. of the* 27^{th} *Annual IEEE Symposium on Foundations of Computer Science*, pages 465–477, 1986.

[KW85] R. M. Karp and A. Wigderson. A fast parallel algorithm for the maximal independent set problem. *Journal of the Association for Computing Machinery*, 32(4):762–773, October 1985.

[Lin87] N. Linial. Locality as an obstacle to distributed computing. In *Proc. 28th IEEE Symp. on Foundations of Computer Science*, pages 331–335, October 1987.

[LM86] C. Leiserson and B. Maggs. Communication-efficient parallel graph algorithms. In *Proc. of International Conference on Parallel Processing*, pages 861–868, 1986.

[Lub86] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal of Comp.*, 15(4):1036–1052, November 1986.

[MR85] G. Miller and J. Reif. Parallel tree contraction and its application. In *Proc. of 26'th Annual IEEE Symposium on Foundations of Computer Science*, pages 478–489, October 1985.

[MST80] D. Matula, Y. Shiloach, and R. Tarjan. *Two Linear-time Algorithms for Five-coloring a Planar Graph.* Technical Report STAN-CS-80-830, Department of Computer Science, Stanford University, Palo Alto, California, November 1980.

[Nao86] J. Naor. *Two Parallel Algorithms in Graph Theory.* Technical Report CS-86-6, Department of Computer Science, The Hebrew University of Jerusalem, Jerusalem, Israel, June 1986.

[PQ82] J. C. Picard and M. Queyranne. A network flow solution to some nonlinear 0-1 programming problems, with applications to graph theory. *Networks*, 12:141–159, 1982.

[Rei85] J. Reif. An optimal parallel algorithm for integer sorting. In *Proc. of 26'th Annual IEEE Symposium on Foundations of Computer Science*, pages 496–503, October 1985.

[Sha86] G. Shannon. *Parallel Independent Set Algorithms for Sparse Graphs.* Technical Report CSD-TR-634, Computer Sciences Dept., Purdue University, October 1986.