ELSEVIER

Information Processing Letters 56 (1995) 333-338

Information Processing Letters

Fast parallel algorithms for the approximate edge-coloring problem

Weifa Liang [1]

Department of Computer Science, Australian National University, Canberra, ACT 0200, Australia

Communicated by F. Dehne; received 1 February 1995

Abstract

This paper presents a parallel algorithm for the following problem: given a simple graph $G(V,E)$, color its edges with $\max\{2^{i-1} * (\lceil \Delta/2^{i-1} \rceil + 2), 2^{i-1} * (\lfloor \Delta/2^{i-1} \rfloor + 3)\}$ colors, $0 \leqslant i \leqslant \lceil \log \Delta \rceil - 1$ such that all edges sharing a common vertex have different colors, where Δ ($= \Delta(G)$) is the maximum vertex degree, and $|V| = n$, $|E| = m$. This algorithm runs in $O((\Delta/2^{i-1})^{3.5} \log^3 \Delta \log n + (\Delta/2^{i-1})^3 \log^4 n)$ time using $O(\max\{n^2, n(\Delta/2^{i-1})^3\})$ processors. Particularly, it only requires $O(\Delta^{1.75} \log^3 \Delta \log n + \Delta^{1.5} \log^4 n)$ time and $O(\max\{n^2, n\Delta^{1.5}\})$ processors when we use $\Delta + \sqrt{\Delta}$ colors to color G's edges. If we use 2.5Δ colors, it only requires $O(\log n \log \Delta)$ time and $O(m)$ processors, based on a CREW PRAM. Unless otherwise specified, our computational model is a COMMON CRCW PRAM in which concurrent read and concurrent write are allowed. In addition, only writing the same value into a memory cell is successful when there exists write conflict.

Keywords: Parallel algorithms; Approximate edge-coloring; Graph problems

1. Introduction

Given a simple graph $G(V,E)$, where $|V| = n$, $|E| = m$. The *edge-coloring* problem is to assign each edge in G a color such that all edges sharing a common vertex have different colors. We wish to use as few colors as possible to color G. The minimum number of colors used is called *chromatic index*, denote by $\chi'(G)$. Let $\Delta(G)$ be the maximum vertex degree in G (Δ for short); obviously $\chi'(G) \geqslant \Delta$. Vizing has already proven that $\Delta \leqslant \chi'(G) \leqslant \Delta + 1$ [9]. In fact, his proof implies $O(nm)$ sequential time algorithm. However, Holyer has shown that deciding whether a graph requires Δ or $\Delta + 1$ colors is NP-complete, even when restricted to the class of cubic graphs [10].

There are a number of parallel algorithms for the edge-coloring problem on special graphs such as planar graphs, bipartite graphs, and Halin graphs [2,4,6-8]. For general graphs, the pioneering work is due to Karloff and Shmoys [1], they presented an edge-coloring parallel algorithm with $\Delta + 1$ colors which requires $O(\Delta^6 \log^4 n)$ time and $O(n^2 \Delta)$ processors based on a COMMON CRCW PRAM. They also presented an randomized NC parallel algorithm with $\Delta + 20\Delta^{1/2+\varepsilon}$ colors for edge-colorings, where $\varepsilon \leqslant 1/4$. Recently, following the idea of Karloff and Shmoys and using some new techniques, Liang, Shen and Hu improved Karloff and Shmoys' algorithm by an $O(\Delta^{2.5})$ factor in time by using $O(\max\{n^2, n\Delta^3\})$ processors on the same model [3]. However whether the edge-coloring problem is in NC is still open.

[1] Email: wliang@cs.anu.edu.au.

In this paper we study the *approximate edge-coloring* problem, that is, we color G's edges with

$$\max\{2^{i-1} * (\lceil \Delta/2^{i-1} \rceil + 2), 2^{i-1} * (\lfloor \Delta/2^{i-1} \rfloor + 3)\}$$

colors, $0 \leqslant i \leqslant \lceil \log \Delta \rceil - 1$, such that all edges sharing a common vertex have different colors. We suggest an efficient algorithm for this problem, based on graph decomposition technique. Our algorithm requires

$$O((\Delta/2^{i-1})^{3.5} \log^3 \Delta \log n + (\Delta/2^{i-1})^3 \log^4 n)$$

time and $O(\max\{n^2, n(\Delta/2^{i-1})^3\})$ processors. The most important is that we have found there exists some kind of trade-off between the number of colors used and the amount of work to assign the colors. That means, the fewer the colors used ($\geqslant \Delta + 1$) to edge-color G, the more time needed for coloring. We show that the approximate edge-coloring can be done in $O(\log n \log \Delta)$ time using $O(m+n)$ processors on a CREW PRAM if we use 2.5Δ colors which implies that this problem is in NC. Meanwhile, we already knew that edge-coloring G with $3\lceil \Delta/2 \rceil$ colors requires $O(\log^3 \Delta n)$ time and $O(n\Delta)$ processors on a CREW PRAM [1]. Particularly, if we use $\Delta + \sqrt{\Delta}$ colors to color G, it only requires

$$O(\Delta^{1.75} \log^3 \Delta \log n + \Delta^{1.5} \log^4 n)$$

time and $O(\max\{n^2, n\Delta^{1.5}\})$ processors on a COMMON CRCW PRAM, compared with the current best result of

$$O(\Delta^{3.5} \log^3 \Delta \log n + \Delta^3 \log^4 n)$$

time and $O(\max\{n^2, n\Delta^3\})$ processors when we use $\Delta + 1$ colors to edge-color G on the same model.

The parallel computational model used in this paper is called COMMON CRCW PRAM that allows concurrent read and concurrent write. In addition, only writing the same value into a memory cell can be successful when write conflict happens.

The remainder of this paper is organized as follows: in Section 2 we will introduce the graph decomposition concept, and then present an efficient algorithm for this decomposition. We then give a brief description of the edge-coloring algorithm with $\Delta + 1$ colors. The approximate edge-coloring algorithm will be given in Section 3. Finally we conclude this paper in Section 4 by introducing an open problem.

2. Preliminaries

2.1. Graph decomposition

Given an Eulerian graph G, the *Eulerian decomposition* is to partition the graph G into edge-disjoint simple cycles. If G is not an Eulerian graph, we add a new vertex v to it, and add an edge between v and each of odd vertices in G so that the resulting graph G' is an Eulerian graph. Let G (or G') be an Eulerian graph. We first partition G into two edge-disjoint subgraphs G_1 and G_2 such that $\Delta(G_1) = \Delta(G_2)$. It can be easily done by traversing an Euler tour and alternatively assigning every edge with *red* and *blue* colors during the Eulerian traversal. Let the subgraph of all red edges be G_1, and the subgraph of all blue edges be G_2. As a result, $\Delta(G_1) = \Delta(G_2)$ and $|E(G_1)| = |E(G_2)|$.

Obviously, for an arbitrary simple graph G, we can partition G into two edge-disjoint subgraphs $G_1(V, E_1)$ and $G_2(V, E_2)$ such that

$$\begin{aligned} &\Delta(G_1) \leqslant \lceil \Delta(G)/2 \rceil \quad \text{and} \\ &\Delta(G_2) \leqslant \lfloor \Delta(G)/2 \rfloor + 1. \end{aligned} \tag{1}$$

However, it is a little more expensive to obtain this partition by Eulerian transversal in parallel computing environments. Here we introduce a more efficient way to partition G into G_1 and G_2. As Furer and Raghavachari noted [11], we do not need to find an Eulerian traversal in G' to achieve the above partition. If we find all edge-disjoint simple cycles in G', then we still can assign the edges on every simple cycle by the color blue and red alternatively. For an even simple cycle, we arbitrarily select a vertex on it as the starting point, and assign each edge red and blue alternatively starting from this point. The odd cycles can be dealt with by applying the simple rule that if two odd cycles C_1 and C_2 sharing a common starting point v, we color C_1 (C_2) by the color red (blue) and blue (red) alternatively, starting from v.

Lemma 1. *Given a graph $G(V, E)$, partitioning G into two edge-disjoint subgraphs $G_i(V, E_i)$, $i = 1, 2$, such that $\Delta(G_1) = \lceil \Delta(G)/2 \rceil$, $\Delta(G_2) = \lfloor \Delta(G)/2 \rfloor + 1$, $E_i \subseteq E$, $E_1 \cap E_2 = \emptyset$, $i = 1, 2$, can be done in* $O(m+n)$ *sequential time, or in* $O(\log n)$ *time using* $O(m+n)$ *processors on the CREW PRAM in which concurrent read is allowed but concurrent write is* *banned, where $|V| = n$ and $|E| = m$.*

Proof. It is easy to prove that this decomposition can be implemented in $O(m+n)$ sequential time, because the time complexity of sequential Eulerian transversal is within this bound. Now we prove Lemma 1 by presenting an algorithm to implement this partition. First of all, we calculate the vertex degree $d(v)$ for every vertex v in graph G. Here we assume that G is an Eulerian graph, if not, we use the above method to make it become an augmented Eulerian graph G'. Then, apply the first part of Atallah and Vishkin's algorithm for finding Euler tour [5] to find all edge-disjoint simple cycles in G or G', which can be implemented in $O(\log n)$ time using $O(m+n)$ processors on a CREW PRAM. Note that here we only need the first part of their algorithm. The dominating computational complexity in their algorithm is of finding a spanning tree which we do not need.

Having done above preprocessing, we now give the partition algorithm. It consists of three steps.

Step 1 is to delete the new vertex and new added edges from the simple cycles if the original graph G is not an Eulerian graph. As a result, the G is partitioned into simple cycles and simple paths.

Step 2 is to color all paths. For each simple path, we select one of its two endpoints as the starting point for coloring. Let P be such a path with two endpoints v and u. Initially we set $T(v) = 0$ for all vertex v in G. If $d(v) = \Delta$ and $d(u) < \Delta$, we select v as the starting point, and color this path with color blue first; If $d(v) = \Delta$ and $d(u) = \Delta$, we select v as the starting point and color this path with blue first. Then we calculate the length of P, if the length of P is even, set $T(u) = 1$. If both $d(v) < \Delta$ and $d(u) < \Delta$, we arbitrarily select one of them to serve as the starting point, and use the color blue first.

Step 3 is to color all simple cycles. For each simple cycle C, regardless of odd or even length, we arbitrarily select a vertex v on it which will be served as the starting point to color, denote by a tuple (v, C). Now we sort all these tuples by key v. After sorting, do 2-coloring on all even cycles with color blue and red alternatively, then delete the tuples corresponding to even cycles from the sorted sequence. It can be implemented by compressing the sequence using the parallel prefix computation [4]. Then the remaining tuples in the sequence correspond to odd cycles. Now we explain what we will do. Let $d(v) = \Delta$ and v be the starting point of k odd cycles, we pair these k odd cycles into $\lfloor k/2 \rfloor$ pairs, and exactly one of them is unpaired. For each pair of cycles, we color one of them by color red first and another with blue first. If k is odd, which color is to be used first for the unpaired cycle is determined by $T(v)$. $T(v) = 1$ means that v is one of two endpoints of a path, and the edge adjacent to v on this path is already colored by red. In order to satisfy the condition (1), the only way we can do now is to color this unpaired cycle with color blue first, otherwise we color this unpaired cycle by color red. The details are as follows:

After having compressed, we calculate the number of times a vertex v is specified as the starting point in these remaining tuples. Denote by $num(v)$ this number. Obviously, all $num(v)$ can be figured out in $O(\log n)$ time using $O(n/\log n)$ processors by prefix computation. Let the length of the remaining tuples be l. For two adjacent tuples (v'_i, C'_i) and (v'_{i+1}, C'_{i+1}), if $v'_i \neq v'_{i+1}$, $num(v'_i)$ is odd, and $T(v'_i) = 1$, we will color C'_i with blue first, delete this tuple from the sequence, $1 \leqslant i \leqslant l-1$. Then we compress the sequence again. Now we color these finally remaining tuples as follows: for each of these remaining tuples, if the rank (or index) of (v, C) in the sequence is odd, we color C by color red first starting from v, otherwise we color C by the color blue first. 2-Coloring a cycle with length at most n can be done in $O(\log n)$ time using $O(n)$ processors, and sorting m elements can be done in $O(\log m)$ time using $O(m)$ processors by the algorithm due to Cole [12]. As a result, G_1 is a subgraph induced by all edges colored red, and G_2 is a subgraph induced by all edges colored blue. It is not difficult to show that the above partition satisfies the conditions

$$\Delta(G_1) \leqslant \lceil \Delta(G)/2 \rceil, \quad \text{and}$$
$$\Delta(G_2) \leqslant \lfloor \Delta(G)/2 \rfloor + 1. \qquad \square$$

2.2. *A brief description of the edge-coloring algorithm*

The edge-coloring problem is to assign G's edges with at most $\Delta + 1$ colors. Here we briefly go through the algorithm by Liang et al. [3], because we will use it later. Their algorithm is basically divided into following three steps. The first step is to edge-color G

with $3\lceil \Delta/2 \rceil$ colors. By this edge-coloring, partition the edge set in G into $\lceil \Delta/2 \rceil$ edge subsets, and each of these subsets is an induced subgraph that contains edge-disjoint simple cycles and paths. The second step is to color these subgraphs using different color sets. The third step is the major step called *merge* step. Let G_A and G_B be two such subgraphs, and their color sets be C_A and C_B. *Merge* these two subgraphs as follows: remove an extra color $\alpha \in C_A \cup C_B$; de-color all edges with α in $G_A \cup G_B$; and re-color the uncolored edges in $G_A \cup G_B$ using the colors in set $C_A \cup C_B - \{\alpha\}$. They first pair these subgraphs, and then do merge in parallel recursively till there is only one remaining subgraph G. In order to do that, they construct several auxiliary graphs to solve the conflict caused by parallelism. The key subroutine in their algorithm is to construct fan graphs for finding the maximal fans by using *directed depth first searching technique*. The details can be found in [3].

Theorem 2 (see [3]). *Given a graph $G(V,E)$ with the maximum vertex degree Δ, edge-coloring G requires*

$$O(\Delta^{3.5} \log^3 \Delta \log n + \Delta^3 \log^4 n)$$

time and $O(\max\{n^2, n\Delta^3\})$ processors on a COMMON CRCW PRAM.

3. The parallel algorithm for approximate edge-colorings

Given $G(V,E)$ with the maximum vertex degree Δ, let G_{2j} and G_{2j+1} be two subgraphs induced from G_j after a partition satisfying the condition (1) in Lemma 1, $1 \leqslant i \leqslant \lceil \log \Delta \rceil - 1$, and $2^{i-1} \leqslant j \leqslant 2^i - 1$. Initially $G_1 = G$. Now we describe an algorithm for approximate edge-coloring G with

$$\max\{2^{s-1} * (\lceil \Delta/2^{s-1} \rceil + 2),\\ 2^{s-1} * (\lfloor \Delta/2^{s-1} \rfloor + 3)\}$$

colors, $0 \leqslant s \leqslant \lceil \log \Delta \rceil - 1$. The algorithm consists of two phases. In the first phase we partition G into a number of edge-disjoint subgraphs and each partition satisfies the conditions of Lemma 1. In the second phase, we apply the edge-coloring algorithm of Liang et al. [3] for every subgraph in the final partition. The detailed algorithm is as follows:

```
Initially $G_1 := G$;
Step 1
  for $i := 1$ to $s - 1$ do
    for $j := 2^{i-1}$ to $2^i - 1$ in parallel do
      partition $G_j$ into two subgraphs $G_{2j}$ and $G_{2j+1}$
    endfor
  endfor

Step 2
  for $i := 2^{s-1}$ to $2^s - 1$ in parallel do
    color $G_i$ with $\max\{\lceil \Delta/2^{s-1} \rceil + 2, \lfloor \Delta/2^{s-1} \rfloor + 3\}$
    colors by the algorithm due to Liang et al. [3]
  endfor.
```

Note that we color the edges of each subgraph by using different color sets. The above algorithm is quite simple. In the following we analyze the number of colors used. Let

$$2^k \leqslant \Delta < 2^{k+1}, \tag{2}$$

then

$$\Delta = a_k 2^k + a_{k-1} 2^{k-1} + \cdots + a_2 2^2 + a_1 2 + a_0, \tag{3}$$

where $a_i = 0$ or $a_i = 1$, $0 \leqslant i \leqslant k$.

Now we definite some functions for later use. Let

$$\phi(i, \Delta) = a_i 2^i, \quad 0 \leqslant i \leqslant \lfloor \log \Delta \rfloor. \tag{4}$$

It is obvious that either $\phi(i,\Delta) = 2^i$ or $\phi(i,\Delta) = 0$, because a_i equals either 0 or 1 by (3). Define

$$\lambda(i,k) = a_i 2^i + a_{i+1} 2^{i+1} + \cdots \\ + a_{k-1} 2^{k-1} + a_k 2^k, \quad 0 \leqslant i \leqslant \lfloor \log \Delta \rfloor \tag{5}$$

and

$$\delta(z) = \begin{cases} 1 & \text{if } z > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Now we rewrite Δ as

$$\Delta = \lambda(0,i) + \lambda(i+1,k), \quad 0 \leqslant i \leqslant k-1. \tag{7}$$

Lemma 3. $2^k \leqslant \lambda(i,k) \leqslant 2^{k-i+1}$, *if $k > 0$ and $i \leqslant k$.*

Lemma 4. $\lambda(i,k) \bmod 2^s = 0$, *for all $0 \leqslant s \leqslant i \leqslant k$ and $k > 0$.*

Lemma 5. *After having s partitions for a graph $G(V,E)$ with the maximum vertex degree Δ. The maximum vertex degree $\Delta(G_j)$ of subgraph G_j in the final partition satisfies the following inequality,*

$$\Delta(G_j) \leqslant \max\{\lceil \Delta/2^s \rceil + 1, \lfloor \Delta/2^s \rfloor + 2\}, \qquad (8)$$

where $2^{s-1} \leqslant j \leqslant 2^s - 1$, $1 \leqslant s \leqslant \lceil \log \Delta \rceil - 1$.

Proof. We prove it by induction. When $s = 1$, by Lemma 1, the inequality (8) holds. Assuming it holds when $s = i$. Now we prove it also holds when $s = i+1$. Let G'' is the subgraph after the ith partitions. By the inductive assumption, we already know that

$$\Delta(G'') \leqslant \max\{\lceil \Delta/2^i \rceil + 1, \lfloor \Delta/2^i \rfloor + 2\}.$$

Now we distinguish two cases:

Case i: $\Delta(G'') = \lceil \Delta/2^i \rceil + 1$, i.e.,

$$\Delta(G'') = \left\lceil \frac{\lambda(0,i-1) + \lambda(i,k)}{2^i} \right\rceil + 1 = \frac{\lambda(i,k)}{2^i} + \delta\left(\frac{\lambda(0,i-1)}{2^i}\right) + 1.$$

By Lemma 1, we now partition G'' into two subgraphs G_1'' and G_2'', which are the subgraphs formed by edges colored red and blue respectively. Therefore,

$$\begin{aligned}\Delta(G_1'') &\leqslant \left\lceil \frac{\Delta(G'')}{2} \right\rceil \\ &= \frac{\lambda(i+1,k)}{2^{i+1}} + \left\lceil \frac{\phi(i,\Delta)}{2^{i+1}} + \frac{\delta(\lambda(0,i-1)/2^i)}{2} + \frac{1}{2} \right\rceil \\ &\leqslant \frac{\lambda(i+1,k)}{2^{i+1}} + 2 \\ &= \max\left\{ \left\lceil \frac{\Delta}{2^{i+1}} \right\rceil + 1, \left\lfloor \frac{\Delta}{2^{i+1}} \right\rfloor + 2 \right\}.\end{aligned}$$

and

$$\begin{aligned}\Delta(G_2'') &\leqslant \left\lfloor \frac{\Delta(G'')}{2} \right\rfloor + 1 \\ &= \left\lfloor \frac{\lambda(i,k)/2^i + \delta(\lambda(0,i-1)/2^i)}{2} \right\rfloor + 1 \\ &= \frac{\lambda(i+1,k)}{2^{i+1}} + \left\lfloor \frac{\phi(i,\Delta)}{2^{i+1}} + \frac{\delta(\lambda(0,i-1)/2^i)}{2} \right\rfloor + 1 \\ &\leqslant \frac{\lambda(i+1,k)}{2^{i+1}} + 2 \\ &= \max\left\{ \left\lceil \frac{\Delta}{2^{i+1}} \right\rceil + 1, \left\lfloor \frac{\Delta}{2^{i+1}} \right\rfloor + 2 \right\}.\end{aligned}$$

Case ii: $\Delta(G'') = \{\lfloor \Delta/2^i \rfloor + 2\}$. The proof is similar to (2) and omitted. □

Theorem 6. *Given a graph $G(V,E)$ with the maximum vertex degree Δ, edge-coloring G with*

$$\lambda(s-1,k) + \max\left\{3 * 2^{s-1}, \left(2 + \delta\left(\frac{\lambda(0,s-2)}{2^{s-1}}\right)\right) * 2^{s-1}\right\}$$

colors can be done in

$$O((\Delta/2^{s-1})^{3.5} \log^3 \Delta \log n + (\Delta/2^{s-1})^3 \log^4 n)$$

time using $O(\max\{n^2, n(\Delta/2^{s-1})^3\})$ *processors on the COMMON CRCW PRAM, where* $0 \leqslant s \leqslant \lceil \log \Delta \rceil - 1$.

Proof. In order to make the maximum vertex degree $\Delta(G_j) \leqslant \max\{\lceil \Delta/2^{s-1} \rceil + 1, \lfloor \Delta/2^{s-1} \rfloor + 2\}$ for every subgraph G_j, $2^{s-1} \leqslant j \leqslant 2^s - 1$, we need $s-1$ partition iterations, and each partition requires $O(\log n)$ time and $O(m+n)$ processors by Lemma 1. So all partitions require $O(s \log n)$ time and $O(m+n)$ processors. After finishing Step 1, there are 2^{s-1} subgraphs with the maximum degree $\Delta(G_j)$, so the number of colors used is

$$\begin{aligned}&\max\left\{2^{s-1} * \left(\left\lceil \frac{\Delta}{2^{s-1}} \right\rceil + 2\right), 2^{s-1} * \left(\left\lfloor \frac{\Delta}{2^{s-1}} \right\rfloor + 3\right)\right\} \\ &\quad \leqslant \lambda(s-1,k) + \max\left\{3 * 2^{s-1}, \left(2 + \delta\left(\frac{\lambda(0,s-2)}{2^{s-1}}\right)\right) * 2^{s-1}\right\},\end{aligned}$$

where $0 \leqslant s \leqslant \lceil \log \Delta \rceil - 1$. □

Corollary 7. *Given a graph $G(V,E)$ with the maximum vertex degree Δ, edge-coloring G with $\Delta + \sqrt{\Delta}$ colors requires*

$$O(\Delta^{1.75} \log^3 \Delta \log n + \Delta^{1.5} \log^4 n)$$

time using $O(\max\{n^2, n\Delta^{1.5}\})$ *processors on a COMMON CRCW PRAM.*

Proof. Let $t = 1/2\lfloor \log \Delta \rfloor - 1$, $t' = \lfloor 1/2 \log \Delta - \log 3 \rfloor$, and let $s = \max\{t, t'\}$. Then the corollary follows from Theorem 6. □

Corollary 8. *Given a graph $G(V, E)$ with the maximum vertex degree Δ, edge-coloring G with 2.5Δ colors requires $O(\log \Delta \log n)$ time and $O(m+n)$ processors on a CREW PRAM.*

Proof. Assign $s = k$ in Theorem 6. In this case, the maximum degree of any subgraph in the final partition is bounded by 3. We color these subgraphs by applying the special algorithm in [1] proposed for these subgraphs instead of applying the algorithm of Liang et al. [3] which can be implemented in $O(\log n)$ time using $O(n)$ processors. The number of colors used is $\lambda(k-1, k) + 3 * 2^{k-1} \leqslant \Delta + 2^k + 1/2 * 2^k \leqslant 2.5\Delta$. □

4. Conclusion

In this paper we study the approximate edge-coloring problem from the parallel viewpoint. By the above discussion, we know that there exists some kind of trade-off between the number of colors used and the time spent to color G's edges. It is a well known open problem whether the edge-coloring problem is in NC. In this paper, we relax the conditions by allowing $\Delta + \sqrt{\Delta}$ colors. Whether this approximate edge-coloring is in NC is unknown.

Acknowledgments

I would like to thank Professor Xiaojun Shen for his many helpful comments and suggestions for improving the presentation of this paper. I also want to thank Dr. Brendan D. McKay and Professor Richard P. Brent for their encouragements and supports for my work.

References

[1] H.J. Karloff and D.B. Shmoys, Efficient parallel algorithms for edge coloring problems, *J. Algorithms* **8** (1987) 39-52.

[2] G.F Lev, N. Pippenger and L.G. Valiant, A fast parallel algorithm for routing in permutation networks, *IEEE Trans. Comput.* **30** (1981) 93-100.

[3] W. Liang, X. Shen and Q. Hu, Parallel algorithms for the edge-coloring and the edge-coloring update problems, *J. Parallel Distributed Comput.* (1995), to appear.

[4] A. Gibbons and W. Rytter, *Efficient Parallel Algorithms* (Cambridge University Press, Cambridge, 1988).

[5] M.J. Atallah and U. Vishkin, Finding Euler tours in parallel, *J. Comput. System Sci.* **29** (1984) 330-337.

[6] A.M. Gibbons, A. Israeli and W. Rytter, Parallel $O(\log n)$ time edge-coloring of trees and Halin graphs, *Inform. Proc. Lett.* **27** (1988) 43-51.

[7] M. Chrobak and M. Yung, Fast algorithms for edge-coloring planar graphs, *J. Algorithms* **10** (1989) 35-51.

[8] M. Chrobak and T. Nishizeki, Improved edge-coloring algorithms for planar graphs, *J. Algorithms* **11** (1990) 102-116.

[9] V.G. Vizing, On an estimate of the chromatic class of a p-graph, *Diskret. Anal.* **3** (1964) 25-30 (in Russian).

[10] I. Holyer, The NP-completeness of edges coloring, *SIAM J. Comput.* **10** (1981) 718-720.

[11] M. Furer and B. Raghavachari, An efficient NC algorithm for finding Hamiltonian cycles in dense directed graphs, in: *ICALP'91, Lectures of Computer Sciences* **510** (Springer, Berlin, 1991) 429-440.

[12] R. Cole, Parallel merge sort, in: *IEEE Symp. on Foundations of Computer Science* (1986).