Very Fast Parallel Algorithms for Approximate Edge Coloring[1]

Yijie Han[2] *Weifa Liang*[3] *Xiaojun Shen*[4]

Abstract

This paper presents very fast parallel algorithms for approximate edge coloring. Let $\log^{(1)} n = \log n$, $\log^{(k)} n = \log(\log^{(k-1)} n)$, and $\log^*(n) = \min\{\ k \mid \ \log^{(k)} n < 1\}$. It is shown that a graph with n vertices and m edges can be edge colored with $(2\lceil \log^{1/4} \log^*(n) \rceil)^c \cdot (\lceil \Delta / \log^{c/4} \log^*(n) \rceil)^2$ colors in $O(\log\log^*(n))$ time using $O(m+n)$ processors on the EREW PRAM, where Δ is the maximum vertex degree of the graph and c is an arbitrarily large constant. It is also shown that the graph can be edge colored using at most $\lceil 4\Delta^{1+4/\log\log\log^*(\Delta)} \log^{1/2} \log^*(\Delta) \rceil$ colors in $O(\log\Delta \log\log^*(\Delta)/\log\log\log^*(\Delta) + \log\log^*(n))$ time using $O(m+n)$ processors on the same model.

Keywords: Analysis of algorithms, graph algorithms, parallel algorithms, edge coloring, PRAM.

1 Introduction

Given a simple graph $G = (V, E)$ with n vertices, m edges and maximum vertex degree Δ, an edge coloring of G is to color the edges of the graph such that no two edges incident to a vertex receive the same color. The minimum number of colors which is sufficient to edge color G is called the *chromatic index* $\chi'(G)$. It is easy to see that $\chi'(G) \geq \Delta$. Vizing [14] has shown that $\chi'(G) \leq \Delta + 1$. Misra and Gries [12] later gave a simple constructive proof of Vizing's theorem. Actually Vizing's proof implies an $O(mn)$ time algorithm for edge coloring a graph with $\Delta + 1$ colors. However, Holyer has shown [6] that deciding whether $\chi'(G) = \Delta$ is *NP*-complete, even when restricted to the class of cubic graphs.

Although there is a polynomial time sequential algorithm for edge coloring a graph with $\Delta + 1$ colors, it is not known whether the problem of edge coloring a graph with $\Delta + 1$ colors is in *NC*. Several parallel algorithms for edge coloring a graph with $\Delta + 1$ colors have been developed [9, 11], but the time complexities of these algorithms are polynomial of Δ. To obtain fast *NC* algorithms, a number of deterministic and randomized parallel algorithms for approximate edge coloring (i.e.

[1]A preliminary version of this paper was presented at 1998 10th IASTED International Conference on Parallel and Distributed Computing and Systems, Las Vegas, Nevada, October 1998.

[2]Computer Science Telecommunications Program, University of Missouri – Kansas City, 5100 Rockhill Road, Kansas, MO 64110, USA. Email: han@cstp.umkc.edu, http://welcome.to/yijiehan

[3]Department of Computer Science, Australian National University, Canberra, ACT 0200, Australia. Email: wliang@cs.anu.edu.au

[4]Computer Science Telecommunications Program, University of Missouri – Kansas City, 5100 Rockhill Road, Kansas City, MO 64110, USA. Email: xshen@cstp.umkc.edu. Research supported in part by NSF grant No. C-CR-9810692.

using more than $\Delta + 1$ colors to edge color G) have also been studied. Fürer and Raghavachari [2] showed that a graph can be edge colored with $\lceil c\Delta \rceil$ colors in $O(\log^2 n)$ time using $O(m+n)$ processors, where c is fixed and $1 < c \leq 2$. They also showed that a graph can be edge colored with Δ^2 colors in $O(\log^*(n))$ time using $O(m+n)$ processors, where $\log^*(n) = \min\{\ k \mid \log^{(k)} n < 1\}$. Liang [10] gave a parallel algorithm which edge colors a graph with 2.5Δ colors in $O(\log n \log \Delta)$ time using $O(m+n)$ processors. Liang's result [10] shows that a graph can be edge colored with $\Delta + \lceil \Delta / \log^c n \rceil$ colors by a parallel algorithm with polylog time complexity, where c is a constant. Karloff and Shmoys [9] presented a randomized NC algorithm with $\Delta + 20\Delta^{1/2+\epsilon'}$ colors for any fixed $\epsilon' < 1/4$. Later Berger and Rompel [1] and Motwani et al [13] proved that the randomness in the algorithm of Karloff and Shmoys can be removed by presenting deterministic NC algorithms for edge-coloring G with $\Delta + \Delta^{1/2+\epsilon}$ colors for any fixed $\epsilon > 0$. Recently, Grable and Panconesi [7] gave another randomized NC algorithm for approximate edge coloring. Their algorithm requires $O(\log \log n)$ time using $O(m\Delta)$ processors with high probability. The number of colors used in their algorithm is $(1+\epsilon)\Delta$ for any $\epsilon > 0$.

In this paper our objective is to edge color a graph as fast as possible. In doing so, one natural question is how many colors needed for this coloring. Formally speaking, not only do an NC algorithm is needed but also is the time complexity of the algorithm kept within $o(\log n)$. The previously known parallel algorithms that satisfy this condition are the algorithm by Fürer and Raghavachari [2] and the algorithm by Grable and Panconesi [7]. Fürer and Raghavachari's algorithm edge colors a graph with Δ^2 colors in $O(\log^*(n))$ time using $O(m+n)$ processors on the EREW PRAM. Grable and Panconesi's algorithm is a randomized algorithm, which uses $O(m\Delta)$ processors and runs in $O(\log \log n)$ time on a CRCW PRAM.

In this paper we first improve upon Fürer and Raghavachari's result by showing that a graph can be edge colored with $(2\lceil \log^{1/4} \log^*(n) \rceil)^c \cdot (\lceil \Delta / \log^{c/4} \log^*(n) \rceil)^2$ colors in time $O(\log \log^*(n))$ using $O(m+n)$ processors on the EREW PRAM, where c is an arbitrarily large constant. The number of colors used is essentially $2^c \lceil \Delta^2 / \log^{c/4} \log^*(n) \rceil$, which can be written as $\lceil \Delta^2 / \log^{c'} \log^*(n) \rceil$ because c is an arbitrarily large constant. Therefore our algorithm uses less colors than the algorithm by Fürer and Raghavachari [2], while our algorithm runs faster. We also show that a graph can be edge colored with $\lceil 4\Delta^{1+4/\log \log \log^*(\Delta)} \log^{1/2} \log^*(\Delta) \rceil$ colors in $O(\log \Delta \log \log^*(\Delta) / \log \log \log^*(\Delta) + \log \log^*(n))$ time using $m+n$ processors on the EREW PRAM.

Without loss of generality, we assume that $V = \{1, 2, ..., n\}$. Every vertex $v \in V$ is associated with an adjacency list $L(v)$ which is stored in an array. Each edge $(u, v) \in E$ is represented twice, once in the adjacency list of u and once in the adjacency list of v. These two representations are linked to each other.

The parallel computational model used is the EREW PRAM [8]. Usually, a parallel algorithm is called *optimal* if $T_p \cdot p = O(T_1)$, where T_p is the time complexity of the parallel algorithm, p is the number of processors used, and T_1 is the time complexity of the best known sequential algorithm. In our case an algorithm is also optimal if $T_p \cdot p = O(m+n)$.

2 Edge Coloring with $(2\lceil \log^{1/4} \log^*(n) \rceil)^c \cdot (\lceil \Delta / \log^{c/4} \log^*(n) \rceil)^2$ Colors

We make use of the following result in the design of our algorithms.

Theorem 1 [4]. A linked list or cycle of size n can be edge colored or vertex colored with 3 colors in $O(\frac{n \log i}{p} + \log^{(i)} n + \log i)$ time using p processors on the EREW PRAM, where i is an adjustable integer parameter. A linked list or cycle of size n can also be colored with $\lceil \log^{(i)} n \rceil$ colors in $O(\log i)$ time using n processors on the EREW PRAM. □

Theorem 1 implies that, to edge color a linked list or cycle with three colors, if $i = \log^*(n)$, it can be done in time $O(\log \log^*(n))$ using n processors. This case is also observed by Beame[3]. Note that the coloring algorithm is an iterative process and therefore if the vertices (edges) of the linked list are initially colored with C colors then the 3 coloring can be done in $O(\log \log^* C)$ time using n processors, see [3][4][5]. If i is a constant, edge coloring a linked list or cycle can be done in time $O(n/p + \log^{(i)} n)$ using p processors.

By applying the scheme outlined in [2], the theorem below follows immediately.

Theorem 2. A graph $G(V, E)$ can be edge colored with Δ^2 colors in $O(\log \log^*(n))$ time using $O(m+n)$ processors on the EREW PRAM.

Proof: We follow the algorithm by Fürer and Raghavachari [2]. First each vertex assigns labels from $\{1, 2, ..., \Delta\}$ to its incident edges so that each incident edge receives a pair of labels. This takes constant time with $O(m)$ processors. Then, each edge has an ordered pair $\langle i, j \rangle$, where $i \leq j$ and i and j are the labels assigned by the two endpoints of the edge. The edge uses $\langle i, j \rangle$ as its temporary color. But the edge coloring obtained may be invalid (an edge coloring is *invalid* if two edges incident to a vertex have the same color). However, for a color $\langle i, j \rangle$ with $i \neq j$, the subgraph induced by the edges colored with $\langle i, j \rangle$ consists of disjoint paths and simple cycles. Now for each such a path or cycle, we first edge color the path or cycle with *three* additional colors $\{\alpha, \beta, \gamma\}$ using Theorem 1. As a result, if an edge receives α, we recolor it with color $\langle i, i \rangle$; if it receives γ, we recolor it with color $\langle j, j \rangle$; if it receives β, we recolor it with color $\langle i, j \rangle$. For a path whose edges are colored with color $\langle i, j \rangle$, we may need to alter the color of the first edge (the edge incident to an endpoint of the path colored with i) and the last edge (the edge incident to another endpoint of the path colored with j) in the path to make sure that the first edge is recolored with $\langle i, i \rangle$ or $\langle i, j \rangle$ and the last edge is recolored with $\langle i, j \rangle$ or $\langle j, j \rangle$. The edge coloring obtained now is a valid edge coloring, which is proven as follows. Let $v_0, v_1, ..., v_t$ be the vertex sequence of a cycle C whose edges are temporarily colored with color $\langle i, j\rangle$. Then after recoloring the edges (v_{k-1}, v_k) and (v_k, v_{k+1}) $(0 \le k \le t,\ v_{-1} = v_t,\ v_{t+1} = v_0)$ are colored with different colors (from $\langle i,i\rangle, \langle i,j\rangle, \langle j,j\rangle$). Any other edge (u, v_k) incident to vertex v_k is colored with a color $\langle x, y\rangle$ where the endpoint of (u, v_k) at vertex u receives label x and the endpoint of (u, v_k) at vertex v_k receives label y. Then $y \ne i, y \ne j$. Therefore if $x \ne i, j$ or edge (u, v_k) is not recolored then the color (x, y) is different than the colors used for coloring and recoloring the edges in cycle C. Now consider the situation where $x = i$ (or j) and edge (u, v_k) is recolored. This happens only when (u, v_k) is the first or the last edge in a path (v_k is an endpoint of the path) colored with the same color. Therefore it is recolored with color (x, y) or (y, y). Thus edge (u, v_k) is also colored with a different color than the colors used for coloring and recoloring the edges in cycle C. Now let $v_0, v_1, ..., v_t$ be the vertex sequence of a path P whose edges are temporarily colored with color $\langle i, j\rangle$. Then after recoloring, the edges incident to internal vertices of the path are recolored with valid colors (this situation is the same as in the case of cycles). Since we recolor edge (v_0, v_1) with $\langle i, j\rangle$ or $\langle i, i\rangle$ and any other edge incident to v_0 is colored with color $\langle x, y\rangle$ where either $x \ne i$ or $x = i$ but $y \ne i$ and $y \ne j$. Therefore after recoloring we obtain a valid edge coloring for v_0. The situation for v_t can be analyzed similarly. We also note that colors (i, i) are possibly assiged to edges in the original temporary coloring. But in this case no two adjacent edges are colored with the same (i, i) color. By Theorem 1, the proposed algorithm has time complexity $O(\log\log^*(n))$ using $O(m + n)$ processors. $\Box$

Remark. If the $O(n/p + \log^{(i)} n)$ time algorithm with 3 colors in Theorem 1 is applied to Theorem 2, then, an optimal parallel algorithm for edge coloring G with Δ^2 colors follows.

If $\Delta = \Omega(\log^{(c)} n)$ for an arbitrarily large constant c, we can prove the following theorem.

Theorem 3. A graph $G(V, E)$ can be edge colored with Δ^2 colors in constant time using $O(m+n)$ processors on the EREW PRAM, if $\Delta = \Omega(\log^{(c)} n)$ for an arbitrarily large constant c.

Proof: As in Theorem 2, first each edge receives two labels i, j. If $i \le j$ then the edge temporarily colors itself with color $\langle i, j\rangle$; otherwise it temporarily colors itself with color $\langle j, i\rangle$. Therefore G is edge-colored by $\Delta(\Delta + 1)/2$ colors. Now if a path or a cycle is colored with the same temporary color $\langle i, j\rangle$, we first color the path or the cycle with $\lceil \log^{(c)} n\rceil < \Delta/2$ colors (from a second set X of colors numbered $1, 2, ..., \lceil \log^{(c)} n\rceil$). By Theorem 1 this can be done in $O(\log c)$ time. Consequently, if an edge on the cycle is colored with the xth color in X, we then recolor it with color $\langle i, x + \Delta\rangle$, $1 \le x \le \lceil \log^{(c)} n\rceil$. This is done for all edges in cycles. For the edges in a path we recolor all edges in the path by the above routine except the first and the last edges. The first and the last edges in the path still retains their original color $\langle i, j\rangle$. For the case in which a path consists of two edges, we then color the first edge with $\langle i, j\rangle$ (i.e. do not change its color) and color the second edge with $\langle j, j\rangle$. Now the number of colors used is no more than Δ^2 (including colors of the form $\langle i, j\rangle$

with $i \leq j$ and colors of the form $\langle i, x+\Delta \rangle$). We claim that the coloring obtained is a valid edge coloring, which is proven as follows.

Let $p_1 = (v_1, v_2, ..., v_t)$ be a simple cycle in which the edges are colored with the same temporary color $\langle i, j \rangle$. If another path or cycle p_2 whose edges are colored with the same temporary color $\langle k, l \rangle$ goes through vertex v_s where $1 \leq s \leq t$ (note that v_s is neither the first nor the last vertex in the path), then we conclude that $k \neq i$, $k \neq j$, $l \neq i$, $l \neq j$. Therefore, after recoloring, the colors used in p_1 and p_2 are disjoint. If an edge incident to v_s is not in any path (or cycle) or it is the first edge in a path p_2 edge colored with the same color, then after recoloring their colors do not change. The color used to color them is $\langle k, l \rangle$ with $k \leq l \leq \Delta$ while the colors used to recolor edges in p_1 are colors $\langle i, x+\Delta \rangle$. Therefore the colors used are disjoint. Finally if an edge incident to v_s is the last edge in a path p_2 edge colored with the same color $\langle k, l \rangle$, this color may be changed to $\langle l, l \rangle$ by recoloring. But the colors used to color p_1 are of the form $\langle i, x+\Delta \rangle$. Therefore the colors used are disjoint. The situation where p_1 is a path can be analyzed similarly. □

In the following we show how to reduce the number of colors used from Δ^2 to $(2\lceil \log^{1/4} \log^*(n) \rceil)^c \cdot (\lceil \Delta / \log^{c/4} \log^*(n) \rceil)^2$ while retaining the $O(\log \log^*(n))$ time complexity, where c is an arbitrarily large constant. Since the algorithm is not straightforward we first explain the main operations used in the algorithm and then give the algorithm.

Initially the edges in the adjacency list of a vertex v is stored consecutively in an array. As our algorithm proceeds, there will be dummies (i.e. empty cells) generated in the adjacency list. When this happens, the edges in the adjacency list of v will still be stored in the array except that there may be dummies scattered among the edges in the adjacency list. We do not require each edge in an adjacency list to be linked to its previous and next edges in the adjacency list. All we require is that edges incident to a vertex are stored in an array with possibly some array cells being dummies. We never compact or clean up the adjacency list by removing dummies because removing dummies will be too costly and also because these dummies do not affect the operations and the correctness of our algorithm. We use list size to indicate the number of memory cells in the array of the adjacency list of a vertex, where one memory cell can be used to store one edge record. When there are no dummies the list size is equal to the degree of the vertex. When there are dummies the list size is larger than the degree of the vertex. The list size of a graph is the maximum value over the list sizes of all vertices. Initially the list size of the graph is Δ. The algorithms we presented before can be interpreted in terms of list size. For example by Theorem 2 a graph with list size L can be colored with L^2 colors in $O(\log \log^* n)$ time using $O(m+n)$ processors, where m is the total number of edges (not the total number of memory cells in the arrays for adjacency lists because we do not need allocate processors for dummies). Of course we have to assume that each edge is associated with a processor. Throughout our algorithm we associate one processor with each edge of the graph. As the algorithm proceeds, edges are moved around in the memory and the processor associated with an edge is always associated with the edge no matter where the edge is moved to.

The initial input of the algorithm is G. The main operations involved in the algorithm are *split, color*, and *combine*, which are explained below.

The split operation is to split each vertex of G into several vertices such that each has list size no more than k, where k is any chosen integer parameter. The split operation simply cuts the array of the adjacency list of each vertex into segments such that each segment contains no more than k consecutive memory cells. The i-th segment contains memory cells from cell $(i-1)k+1$ to cell ik. The last segment may contain less than k memory cells. Each segment is the adjacency list of a new vertex u'. Suppose edge (u, v) in the adjacency list of u is in segment u' after cutting and the edge (v, u) in the adjacency list of v is in segment v' after cutting, then edge (u, v) in the original graph becomes edge (u', v') in the new graph after splitting. If the split operation is well understood, it is not difficult to see that splitting takes constant time on the EREW PRAM if $O(m)$ processors are available, where m is the number of edges. Note that m is not the total list size of all vertices. Denote by G_1 the graph after the split operation. Since the list size of G_1 is k, the degree of $G_1 \leq k$.

The color operation is to color graph G_1 (of list size k) with $2k-1$ colors which is done as follows. First color G_1 with k^2 colors by using Theorem 2. This takes $O(\log\log^* n)$ time with $O(m)$ processors, where m is the number of edges of G_1. Then the algorithm consists of k^2 phases. In phase j we work on the edges colored with color j. All these edges form a matching in G_1. If edge (u, v) is colored with color j, (u, v) will use $d(u) + d(v) - 2$ processors to recolor (u, v), where $d(u)$ and $d(v)$ are the degrees of u and v in G_1, respectively. Note that we only use a total of $2m$ processors, where m is the number of edges. We cannot use $list\_size(u) + list\_size(v) - 2$ processors for edge (u, v), where $list\_size(u)$ is the list size of u, for if we did, we would need more than $O(m)$ processors. We use these processors to find an unused color among the first $d(u) + d(v) - 1$ colors. We do this by first allocating an array A of size k^2. Then $d(u) + d(v) - 2$ processors write the colors used for coloring edges incident to u and v (except the color used for coloring (u, v)) into A. If color i is used a bit 1 will be written into the i-th cell of A. We now imagine a balanced binary tree built on array A. The array cells are the leaves of the binary tree. The $d(u) + d(v) - 2$ processors are initially associated with the leaves having value 1's. These processors then climb up the tree, one level in a substep, for $O(\log k)$ substeps. As a processor climbing up the tree if it finds the sibling node is a dummy it carries this information with it. If two processors meet at a node of the tree, one processor will stop climbing and the other processor continues climbing the tree. When the root of the tree is reached an unused color among the first $d(u) + d(v) - 1$ colors is found. We then use this unused color to recolor edge (u, v). It is easy to see that this scheme uses a total of $O(m)$ processors for the whole graph, and it can be implemented on the EREW PRAM. Each phase takes $O(\log k)$ time with $O(m)$ processors, where m is the number of edges. After these k^2 phases we reduce the number of colors used to $2k-1$. The total time expended is $O(\log\log^* n + k^2 \log k)$ with $O(m+n)$ processors.

The combine operation is to combine edges colored with the same color together. Suppose a vertex v in G is split into $v_1, v_2, ..., v_t$ vertices in G_1 after the split operation and each v_i has list size no more than k. After the color operation edges are colored with $2k-1$ colors. If the original graph has list size L then $t < \lceil L/k \rceil$. In the combine operation we create a new graph G_2. The combine operation is to combine all edges in $v_1, v_2, ..., v_t$ colored with color j into the adjacency list of a newly created vertex in G_2. Because there are $2k-1$ colors we create $2k-1$ new vertices $u_1, u_2, ..., u_{2k-1}$ in G_2 for the original vertex v in G. New vertex u_i will have the edges colored i in vertices $v_1, v_2, .., v_t$ incident to it. Vertex u_i uses an array of size t to store its adjacency list. Edge colored i incident to vertex v_j of G_1 moves itself into the j-th memory cell of new vertex u_i of G_2. This is done by indexing and takes constant time. If no edge incident to vertex v_j is colored i then the j-th memory cell of the new vertex u_i contains a dummy. This is where dummy gets generated. After executing the combine operation we have a new graph G_2 with list size $\lceil L/k \rceil$. G_2 consists of $2k-1$ subgraphs. Subgraph i contains the edges colored with color i. Subgraph i and subgraph j are not connected if $i \neq j$. After combine operation vertex v in G is divided into $2k-1$ vertices $u_1, u_2, ..., u_{2k-1}$ in G_2, where vertex u_i is in subgraph i. The list size of each vertex u_i is no more than $\lceil L/k \rceil$.

Suppose we have a coloring scheme to edge color G_2 with C colors. Then we obtain a coloring with $C \cdot (2k-1)$ colors for G. Because if edge (x, y) in G becomes edge (x_i, y_i) in G_2 (note that the two subscripts are equal because (x, y) is colored with color i in the color operation and therefore edge (x, y) moves itself to the i-th subgraph of G_2 in the combine operation) and receives color z among the C colors, we can recolor it with color $(i-1)C + z$. We thus obtain a valid coloring for G with $C \cdot (2k-1)$ colors.

Note that G_2 is a graph with list size $\lceil L/k \rceil$. To color G_2 we can recursively or iteratively apply the split, color and combine operations. Each application of these operations reduces the list size of the graph by roughly a factor of k. Thus we can reduce the list size of the graph until it is small enough and then we use Theorem 2 to color the resulting graph.

Now we estimate how much memory is used in the algorithm. The representation of G uses cm memory cells for m edges, where c is a constant. Consider G_1, each vertex in G_1 has list size k. Therefore there are roughly m/k vertices in G_1. The construction of G_1 uses ckm memory because k^2 memory cells is used for each vertex in G_1 and there are about m/k vertices in G_1. However, G_1 is not needed after G_2 is constructed. Thus memory used for G_1 can be reclaimed.

Each vertex v in G is divided into $2k-1$ vertices $u_1, u_2, ..., u_{2k-1}$ in G_2. Each u_i has list size $\lceil list\_size(v)/k \rceil$, where $list\_size(v)$ is the list size of v in G. Thus the amount of memory used for v is $(2k-1) * \lceil list\_size(v)/k \rceil$ which is roughly $2 * list\_size(v)$. Thus the memory used for G_2 is roughly $2cm$. Thus the memory used for storing the adjacency lists of the input graph for the next level of recursion is roughly $2cm$. This indicates that each level of recursion (of calling split, color and combine) doubles the size of the memory. Of course the increased memory are used for storing dummies. If we invoke d levels of recursion the amount of memory used for storing the adjacency lists will be $O(2^d m)$. Note that d can be a function of n instead of a constant. Note also that although the amount of memory used is doubled for each level of recursion, the list size of the graph is reduced by a factor of k for each level of recursion. We can go to the extreme, i.e. invoke sufficient number of levels of recursion to reduce the list size of the graph to a constant. That is exactly what we will do in Section 3. Note also that throughout the algorithm we use only $O(m+n)$ processors. There is one processor allocated and associated with each edge. As the edge moves itself around in the memory the allocated processor is always associated with it. No processors are allocated for dummies.

Now we are ready to give the algorithm as follows.

Algorithm **Edge_Color**(G, L, x)
/* G is the input graph. L is the list size of G. x is an integer parameter. */

Step 1. If $L = 0$ then **return**.
Otherwise, if $L = 1$ then edge color G with 1 color; **return**.
If $x = 0$ then edge color G with L^2 colors by applying Theorem 2; **return**.

Step 2. Split each vertex v of G into at most $\lceil L/\log^{1/4}\log^*(n)\rceil$ new vertices $v_1, v_2, ..., v_{\lceil L/\log^{1/4}\log^*(n)\rceil}$ such that each vertex v_i has degree $\lceil \log^{1/4}\log^*(n)\rceil$ (with the possibility that the last vertex has degree $< \lceil \log^{1/4}\log^*(n)\rceil$). Denote by G_1 this transformed graph. This step implements the split operation with $k = \lceil \log^{1/4}\log^*(n)\rceil$.

Step 3. Edge color G_1 with $O(\log^{1/2}\log^*(n))$ colors in time $O(\log\log^*(n))$ using Theorem 2. Note that the list size of G_1 is no more than $\lceil \log^{1/4}\log^*(n)\rceil$.

Step 4. We proceed on G_1 in $O(\log^{1/2}\log^*(n))$ phases sequentially, each phase corresponds to a color in the color set $\{1, 2, \ldots, O(\log^{1/2}\log^*(n))\}$. In phase j we work on color j. All the edges of G_1 colored with color j form a matching. For each edge (u, v) of G_1 colored with color j, the edge (u, v) checks which color among colors $\{1, 2, ..., d(u) + d(v) - 1\}$, is not used for coloring the edges (except (u, v)) incident to its two endpoints. Since there are $d(u) + d(v) - 2$ edges besides (u, v) which are incident to u and v, there is at least one such a free color. Then, (u, v) recolors itself using this color. Thus, after sequencing through the $O(\log^{1/2} \log^*(n))$ colors we reduce the number of colors used to $2\lceil \log^{1/4} \log^*(n) \rceil - 1$. Steps 3 and 4 implement the color operation.

Step 5. First, combine the vertices $v_1, v_2, ..., v_{\lceil L/\log^{1/4} \log^*(n) \rceil}$ of G_1 into v of G. That is, transform G_1 back to G. Group edges incident to v colored with the same color together. There are at most $\lceil L/\log^{1/4} \log^*(n) \rceil$ edges incident to v colored with the same color. Then, transform G into G_2, i.e, divide each vertex v of G into $2\lceil \log^{1/4} \log^*(n) \rceil - 1$ vertices $u_1, u_2, ..., u_{2\lceil \log^{1/4} \log^*(n) \rceil - 1}$ such that all edges incident to v which are colored with the same color i are incident to vertex u_i. Note that the list size of G_2 is no more than $\lceil L/\log^{1/4} \log^*(n) \rceil$. In summary, the computation involved in this step is to allocate $\lceil L/\log^{1/4} \log^*(n) \rceil$ memory cells to each u_i and to move an edge incident to v_j colored with color i into the j-th memory cell of u_i. Thus this step takes constant time. This step implements the combine operation.

Step 6. Edge color G_2 by calling Edge_Color(G_2, $\lceil L/\log^{1/4} \log^*(n) \rceil$, $x - 1$). Let C be the number of colors used in this step.

Step 7. Group the vertices $u_1, u_2, ..., u_{2\lceil \log^{1/4} \log^*(n) \rceil - 1}$ of G_2 back to v of G. That is, transform G_2 back to G. If edge e incident to u_i is colored with color z in Step 6, we recolor it with color $(i - 1) \cdot C + z$. It is not hard to see that the color obtained for G is a valid edge coloring.

Theorem 4. A graph $G(V, E)$ can be edge colored with $(2\lceil \log^{1/4} \log^*(n) \rceil)^c \cdot (\lceil \Delta/\log^{c/4} \log^*(n) \rceil)^2$ colors in $O(\log \log^*(n))$ time using $O(m + n)$ processors, where c is an arbitrarily large constant.

Proof: First, Edge_Color uses $2\lceil \log^{1/4} \log^*(n) \rceil - 1$ colors to edge color G_1. Then, use this edge coloring, transform G into G_2, and edge color G_2 recursively. Finally, the edge coloring of G is derived from the edge coloring of G_2 in Step 7. The correctness of algorithm Edge_Color can be easily verified.

If the initial call is Edge_Color(G, Δ, c), the recursion will proceed to c levels. Let $N(c)$ be the total number of colors used for edge coloring G, and $\mathit{list\_size}(c)$ be the list size of the graph obtained at level c. Then, if $\mathit{list\_size}(c) \geq \lceil \log^{1/4} \log^*(n) \rceil$, we have $N(c) \leq N(c-1) \cdot (2\lceil \log^{1/4} \log^*(n) \rceil - 1)$ and $\mathit{list\_size}(c - 1) \leq \mathit{list\_size}(c)/\lceil \log^{1/4} \log^*(n) \rceil$. Thus, $\mathit{list\_size}(0) \leq \lceil \Delta/\log^{c/4} \log^*(n) \rceil$ and $N(c) \leq N(0) \cdot (2\lceil \log^{1/4} \log^*(n) \rceil - 1)^c \leq (\mathit{list\_size}(0))^2 \cdot (2\lceil \log^{1/4} \log^*(n) \rceil - 1)^c \leq (2\lceil \log^{1/4} \log^*(n) \rceil)^c \cdot (\lceil \Delta/\log^{c/4} \log^*(n) \rceil)^2$.

Assume that there are $O(m + n)$ processors available. Then, Step 1 takes $O(\log \log^*(n))$ time. Step 2 takes constant time. Step 3 takes $O(\log \log^*(n))$ time. Step 4 takes

$O(\log^{1/2}\log^*(n)\log\log\log^*(n))$ time. In each phase of Step 4, the edges colored with the same color form a matching, and an edge (u,v) in the matching can be re-colored using $d_{G_1}(u)+d_{G_1}(v)-2$ processors. We use $O(\log\log\log^*(n))$ time to pick the color in each phase within Step 4. In Step 5 we treat the edge colored with i which is incident to v_j as the j-th edge of u_i. Therefore, the combine operation in Step 5 can be done in constant time. Step 7 takes constant time. Let $T(c)$ be the time taken by Edge_Color(G, Δ, c). We then have $T(c)\le T(c-1)+O(\log\log^*(n))$. Since $T(0)=O(\log\log^*(n))$ and c is a constant, $T(c)=O(\log\log^*(n))$. □

3 Edge Coloring with $\lceil 4\Delta^{1+4/\log\log\log^*(\Delta)}\log^{1/2}\log^*(\Delta)\rceil$ Colors

If we invoke Edge_Color(G, Δ, x) with $x=\lceil 4\log\Delta/\log\log\log^*(n)\rceil$, then the graph obtained at the xth level of recursion is of constant list size. Because each level of recursion contributes a factor of $\lceil 2\log^{1/4}\log^*(n)\rceil$ colors, the number of colors used for the edge-coloring of G is

$$\begin{aligned}
&(2\log^{1/4}\log^*(n))^{\lceil 4\log\Delta/\log\log\log^*(n)\rceil+1}\\
&\le (2\log^{1/4}\log^*(n))^{4\log\Delta/\log\log\log^*(n)+2}\\
&= (\log^{1/4}\log^*(n))^{4\log\Delta/\log\log\log^*(n)}\cdot 2^{4\log\Delta/\log\log\log^*(n)}\cdot(2\log^{1/4}\log^*(n))^2\\
&= \Delta\cdot\Delta^{4/\log\log\log^*(n)}\cdot 4\log^{1/2}\log^*(n)\\
&= 4\Delta^{1+4/\log\log\log^*(n)}\log^{1/2}\log^*(n).
\end{aligned}$$

Thus, the number of colors used is less than Δ^2 when Δ is relatively large. In fact, the $4\log^{1/2}\log^*(n)$ factor comes from our analysis. The real number of colors used is always less than Δ^2. The time complexity of the algorithm becomes $O(\log\log^*(n))\cdot O(\log\Delta/\log\log\log^*(n))$ $=O(\log\Delta\log\log^*(n)/\log\log\log^*(n))$ because each level of recursion adds an item of $O(\log\log^*(n))$ in the time complexity, and this time complexity is less than $O(\log n)$ when Δ is small. The total number of processors used is $O(m+n)$. Therefore, we have

Theorem 5. A graph $G(V,E)$ can be edge colored with $\lceil 4\Delta^{1+4/\log\log\log^*(n)}\log^{1/2}\log^*(n)\rceil$ colors in time $O(\log\Delta\log\log^*(n)/\log\log\log^*(n))$ using $O(m+n)$ processors on the EREW PRAM. □

Note that the result in Theorem 5 can be improved further. The observation is as follows. During the precomputation, if we color the vertices of the input graph with 3^{Δ^2} colors, then the three coloring of a linked list in Theorem 1 can be done in $O(\log\log^*(3^{\Delta^2}))=O(\log\log^*\Delta)$ time instead of $O(\log\log^* n)$ (see the paragraph immediately below Theorem 1). Therefore the time complexity in Theorem 2 also becomes $O(\log\log^*\Delta)$ instead of $O(\log\log^* n)$. Also we can split v into $\lceil L/\log^{1/4}\log^*(\Delta)\rceil$ vertices rather than $\lceil L/\log^{1/4}\log^*(n)\rceil$ vertices in Step 2 of algorithm Edge_Color. Then the graph G_1 in Step 3 of algorithm Edge_Color has list size $\lceil\log^{1/4}\log^*(\Delta)\rceil$. Step 3 can now be done in $O(\log\log^*\Delta)$ time. Step 4 of algorithm Edge_Color now has only

$O(\log^{1/2}\log^* \Delta)$ phases and takes no more than $O(\log\log^* \Delta)$ time. Step 5 and Step 7 still take constant time. Thus each recursion level of algorithm Edge_Color takes $O(\log\log^* \Delta)$ time. Also each level of recursion now reduces the list size by a factor of $\lceil \log^{1/4}\log^* \Delta\rceil$ in the proof of Theorem 4. Therefore the total number of recursion levels needed to reduce the list size of the input graph to constant is $\lceil 4\log\Delta/\log\log\log^* \Delta\rceil$. This will give time complexity $O(\log\Delta\log\log^*\Delta/\log\log\log^*\Delta)$ for algorithm Edge_Color. Also each level of recursion now increases the number of colors used by a factor of $2\lceil\log^{1/4}\log^*\Delta\rceil-1$. The total number of colors used becomes $(2\log^{1/4}\log^*\Delta)^{\lceil 4\log\Delta/\log\log\log^*\Delta\rceil+1}$ $\leq \lceil 4\Delta^{1+4/\log\log\log^*\Delta}\log^{1/2}\log^*\Delta\rceil$.

Now we deal with the precomputation. We first use Theorem 2 to edge color the input graph with Δ^2 colors. Then each edge arbitrarily labels one of its endpoints with 1 and the other one with 2. Then each vertex v colors itself with color $(a_1, a_2, \ldots, a_{\Delta^2})$, where $a_i = 1$ if there is an edge incident to v colored with i and v receives label 1; $a_i = 2$ if there is an edge incident to v colored with i and v receives label 2; $a_i = 0$ if there is no edge incident to v colored with i. The vertex coloring we obtained is a valid vertex coloring. The number of colors used is 3^{Δ^2}. The vertex color can be computed as $\sum_{i=1}^{\Delta^2} a_i 3^{i-1}$. Note that if $a_i = 0$ then the term $a_i 3^{i-1}$ need not to be evaluated. Thus the total number of processors used for vertex coloring can be reduced to $O((m+n)/\log\Delta)$. This method of computing the vertex coloring is presented to us by a referee. Although our original method uses $O(\log\Delta)$ time it requires $O(m+n)$ processors. We need $O(\log\log^*(n))$ time for the edge coloring and another $O(\log\Delta)$ time for the vertex coloring from the edge coloring. Thus, the time complexity for the precomputation is $O(\log\log^*(n) + \log\Delta)$.

Theorem 6. A graph can be edge colored with $\lceil 4\Delta^{1+4/\log\log\log^*(\Delta)}\log^{1/2}\log^*(\Delta)\rceil$ colors in time $O(\log\Delta\log\log^*(\Delta)/\log\log\log^*(\Delta)+\log\log^*(n))$ using $O(m+n)$ processors on the EREW PRAM. $\square$

4 Conclusions

We presented very fast parallel algorithms for approximate edge coloring. It is not known whether the number of colors can be reduced further while retaining the $O(\log\log^*(n))$ time. We could also ask whether the number of colors can be reduced further while retaining polylog of Δ time.

Acknowledgment

The authors thank the referees for their comments and suggestions which improved the paper in many ways. In particular one referee found a mistake in the original version and he/she also suggested a better vertex coloring method for Theorem 6 which we wrote into this paper.

References

[1] B. Berger and J. Rompel. Simulating ($\log^c n$)-wise independence in NC. *J. of the ACM* **38** (1991), 1026–1046.

[2] M. Fürer and B. Raghavachari. Parallel edge coloring approximation. *Parallel Processing Letters* **6** (1996), 321–329.

[3] A. V. Goldberg, S. A. Plotkin, G. E. Shannon. Parallel symmetry-breaking in sparse graphs, *SIAM J. on Discrete Math.*, Vol 1, No. 4, 447-471(Nov., 1988).

[4] Y. Han. Matching partition a linked list and its optimization. *Proc. 1st Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'89)*, 1989, 246–253.

[5] Y. Han. An optimal linked list prefix algorithm on a local memory computer. *Proc. 1989 Computer Science Conference (CSC'89)*, 278-286(Feb., 1989).

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

[7] D. A. Grable and A. Panconesi. Nearly optimal distributed edge coloring in $O(\log\log n)$ rounds. *Proc. 8th Annual ACM–SIAM Symp. on Discrete Algorithms*, January, 1997, 278–285.

[8] J. JáJá. *An Introduction to Parallel Algorithms.* Addison-Wesley, 1992.

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

[10] W. Liang. Fast parallel algorithms for the approximate edge-coloring problem. *Inform. Process. Lett.* **56** (1995), 333–338.

[11] W. Liang, X. Shen, and Q. Hu. Parallel algorithms for the edge-coloring and edge-coloring update problems. *J. of Parallel and Distributed Computing* **32** (1996), 66–73.

[12] J. Misra and D. Gries. A constructive proof of Vizing's theorem. *Inform. Process. Lett.* **41** (1992), 131–133.

[13] R. Motwani, J. Naor and M. Naor. The probabilistic method yields deterministic parallel algorithms. *J. of Computer and System Sci.* **49** (1994), 478–516.

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