1992

Parallel Algorithms for Gray-Scale Digitized Picture Component Labeling on a Mesh-Connected Computer

Susanne E. Hambrusch
Purdue University, seh@cs.purdue.edu

Xin He

Russ Miller

Report Number:
92-007

PARALLEL ALGORITHMS FOR GRAY-SCALE DIGITIZED PICTURE COMPONENT LABELING ON A MESH-CONNECTED COMPUTER

Susanne Hambrusch
Xin He
Russ Miller

CSD-TR-92-007
January 1992

Parallel Algorithms for Gray-Scale Digitized Picture Component Labeling on a Mesh-Connected Computer *

Susanne Hambrusch† Xin He‡ Russ Miller§

Abstract

We consider the problem of labeling connected components in a gray-scale image so that every component is connected, the maximum difference in the gray-scale values of the pixels within any component does not exceed a given value, and no component can be merged with a neighboring component. We develop two asymptotically optimal $\Theta(n)$ time algorithms for generating such labelings on a mesh-connected computer when the image is mapped onto the mesh with one pixel per processor. The first algorithm operates directly on the image and is based on a divide-and-conquer approach. Although it is simple, it has the potential drawback of possibly assigning two adjacent pixels with the same gray-scale value to different components. The second algorithm avoids this potential drawback. It works with a graph representation of the image and it allows larger components to be formed from smaller ones by taking into account properties and characteristics of the smaller components.

Keywords: Connected components, gray-scale images, mesh-connected computers, parallel algorithms.

---

*This research was partially supported by National Science Foundation grants IRI-8800514, IRI-9108288, CCR-9011214, Office of Naval Research grants N00014-84-K-0501 and N00014-86-K-0689, and DARPA contract DABT63-92-C-0022. A preliminary version of this paper appeared in the *4-th ACM Symposium on Parallel Algorithms and Architectures*, 1992.

†Dept. of Comp. Sci., Purdue University, West Lafayette, IN 47907; (317) 494-1831; seh@cs.purdue.edu

‡Dept. of Comp. Sci., 226 Bell Hall, SUNY-Buffalo, Buffalo, NY 14260; (716) 636-3199; xinhe@cs.buffalo.edu

§Dept. of Comp. Sci., 226 Bell Hall, SUNY-Buffalo, Buffalo, NY 14260; (716) 636-3295; miller@cs.buffalo.edu

1 Introduction

Determining the connected components of a digitized picture is a fundamental problem in image processing and computer vision. Most previous work on connectivity focused on different forms of connectivity for binary images (c.f., [5, 10, 11, 12, 16, 18] and references contained therein). Real applications produce gray-scale images and a standard practice is to transform gray-scale images into binary images through thresholding. Relevant and crucial information is often lost during thresholding. In this paper, we consider definitions and algorithms for gray-scale connectivity. We introduce a definition of connectivity for gray-scale images that (*i*) allows pixels in the same component to have different gray-scale values with respect to certain input parameters and (*ii*) requires components to satisfy a maximal property.

Given this definition of connectivity, we present two asymptotically optimal algorithms to solve the gray-scale component labeling problem on the mesh architecture. The mesh architecture is well-suited for solving problems on images. Further, since transporting algorithms designed for other architectures onto the mesh generally leads to inefficient algorithms, the design of parallel algorithms tailored towards the mesh architecture remains an important task in parallel processing. We assume that the input is an $n \times n$ gray-scale image mapped one pixel per processor onto an $n \times n$ mesh-connected computer. In addition to the image, we are given two input parameters, the *range parameter* ϵ and the *adjacency parameter* δ. Our algorithms label the components so that

- every component is connected,
- the maximum difference in the gray-scale values of the pixels within any component does not exceed the range parameter ϵ, and
- no component can be merged with a neighboring component.

Our two algorithms have significantly different characteristics. The algorithm presented in Section 2 is a divide-and-conquer algorithm. It determines the components directly from the image, where the combining of subsolutions is of a systolic nature. The algorithm is relatively straightforward and simple, but has two potential drawbacks. First, it can assign two adjacent pixels with the same gray-scale value to different components. Secondly, components get merged in a greedy fashion and this does not allow for incorporating decisions on merging components according to properties other than the range parameter. The algorithm presented in Section 4 avoids these potentially undesirable features and allows more flexibility on how components get merged. This second algorithm does not work with the image. Instead, it works with a graph representation of the image. It exploits the ability

of the mesh to efficiently determine a special type of independent set of a planar graph, as presented in Section 3. The algorithm presented in Section 3 is also an essential building block for designing optimal mesh algorithms for solving the maximal independent set and the 5-coloring problem for planar graphs. These results are described in an Appendix.

The remainder of this introductory section gives the definitions used throughout the paper. Let D be an $n \times n$ gray-scale digitized picture. Without loss of generality, we assume $n = 2^k$. Each pixel $D(i,j)$ can take on values $[0,\ldots,g]$, where $g \leq c \log n$, for some constant c. The values represent shades of gray, with 0 representing white (i.e., the background color), and g representing black. Using the definition of 4-connectivity, we say that pixel $D(i,j)$ has four *neighbors*, namely, $D(i+1,j)$, $D(i-1,j)$, $D(i,j+1)$, and $D(i,j-1)$, assuming they exist. If two neighboring pixel x and y contain non-zero values whose difference is at most δ, then x and y are *δ-adjacent.* Recall that the adjacency parameter δ is given as input. A *(connected) component* U is a set of pixels of D such that for all pairs of pixels $p_1, p_2 \in U$, there exists a path $p_1 = x_1, x_2, \ldots, x_k = p_2$ of pixels such that $x_i \in U$, $1 \leq i \leq k$, $k \leq n^2$, and x_i and x_{i+1} are δ-adjacent in D, $1 \leq i \leq k-1$. We define the *range of* U to be $R(U) = [a_U, b_U]$, where a_U is the minimum value, and b_U is the maximum value, among the pixels in U. Given the range parameter ϵ, we say that a component U is *valid* if and only if $b_U - a_U \leq \epsilon$. A solution to the gray-scale component labeling problem assigns a label to every non-zero pixel so that each set of pixels with the same label is a valid component. Observe that in a valid labeling we have $\delta \leq \epsilon$. Furthermore, the labeling has to satisfy the maximal property; i.e., no valid component can be merged with a neighboring valid component to form a (larger) valid component.

Our definition of connectivity for gray-scale images does not result in unique labelings. For example, Figure 1 shows two possible maximal labelings for a given 8×8 image. The algorithm described in Section 4 uses this feature of the definition to its advantage. Whenever the algorithm has a choice in how to merge components, it can use properties and characteristics of the components to be merged to guide the decision-making process. Notice that our definition of connectivity is more meaningful in some situations than others. As in the case of thresholding, for certain images relevant information may not be represented in the connected components generated.

The model of computation used in this paper is the mesh-connected computer. The *(2-dimensional) mesh-connected computer (mesh) of size* n^2 is an SIMD machine with n^2 simple *processing elements (PEs)* arranged in a square lattice. For all $i,j \in [0,\ldots,n-1]$, let $P_{i,j}$ represent the PE in row i and column j. Processor $P_{i,j}$ is connected via bidirectional unit-time communication links to its four *neighbors*, $P_{i-1,j}$, $P_{i+1,j}$, $P_{i,j-1}$, and $P_{i,j+1}$, assuming they exist. Each PE has a fixed number of registers (words), each of size $O(\log n)$, and can perform standard arithmetic and Boolean operations on the contents of these registers in

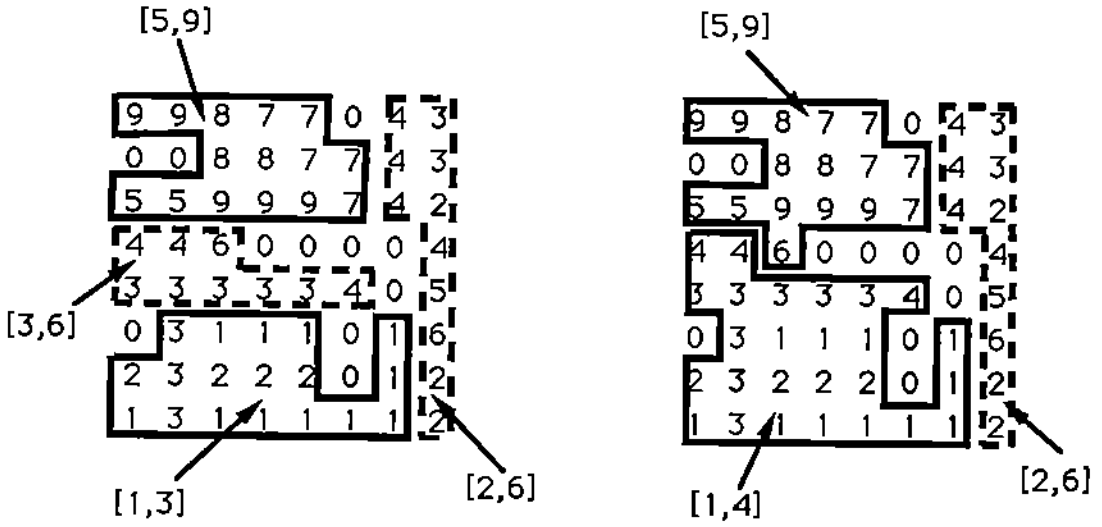


Figure 1: Two maximal labelings for a gray-scale image with $\epsilon = \delta = 4$.

unit time. Each PE can also send or receive a word of data to or from each of its neighbors in unit time. Each PE contains its row and column indices, as well as a unique identification register, the contents of which is initialized to the PE's row-major index, shuffled row-major index, snake-like index, or proximity order index [11]. (If necessary, these values can be generated in $\Theta(n)$ time.)

The communication diameter of a mesh of size n^2 is $\Theta(n)$, as can be seen by examining the distance between PEs in opposite corners of the mesh. This means that if a PE in one corner of the mesh requires data from a PE in another corner of the mesh at some time during an algorithm, then $\Omega(n)$ is a lower bound on the running time of the algorithm. Since we assume that the input is such that processor $P_{i,j}$ contains pixel $D(i,j)$, it is easy to see that, based on the communication diameter, the problem considered in this paper has a lower bound of $\Omega(n)$ on the running time.

In this paper, we will frequently use $\Theta(n)$ time *standard mesh operations* such as sorting, random access read, random access write, compression, and parallel prefix. The reader is referred to [10, 11, 13, 14, 20], and the references contained therein, for complete descriptions, algorithms, and analyses of these operations.

2 A Divide-and-Conquer Algorithm

In this section, we present our divide-and-conquer algorithm for solving the gray-scale component labeling problem. We first describe a $\Theta(n \log n)$ time algorithm which is based on row operations. (Note that existing meshes, such the Connection Machine CM-2, are often designed to exploit such operations.) We then show how to obtain, through minor modifications of this algorithm, an optimal $\Theta(n)$ time algorithm. Both algorithms determine a maximal labeling by merging maximal labelings of subimages in $\Theta(\log n)$ iterations.

The $\Theta(n \log n)$ time algorithm consists of two phases. In the first phase, every row i of the mesh determines, in $\Theta(n)$ time, a maximal gray-scale labeling for the pixels in row i, $0 \leq i \leq n-1$. This is done using a greedy strategy processing the pixels from left to right. A pixel is included into the currently formed component as long as validity is maintained. Otherwise, a new component is initialized. In order for the second phase of the algorithm to work correctly, we require that the component numbers assigned in different rows are distinct. This can easily be achieved by using, for example, the indices of the processors as the component numbers. At the end of this first phase, every non-zero pixel knows its row-restricted component number and range.

The second phase merges the maximal labelings obtained for the n rows in $\log_2 n$ stages, with each stage taking $\Theta(n)$ time. During the i-th stage, we use the information about components having pixels in rows $2^i j - 2^{i-1} - 1$ and $2^i j - 2^{i-1}$ to determine a maximal labeling for the image induced by the pixels in the 2^i rows having indices $2^i j - 2^i, 2^i j - 2^i + 1, \ldots, 2^i j - 1$, $1 \leq j \leq n/2^i$, $1 \leq i \leq \log_2 n$. We refer to the labeling generated by stage i as the *i-restricted labeling*. (The 0-restricted labeling corresponds to the row-restricted labeling generated in the first phase.) Let $r = 2^i j - 2^{i-1} - 1$. During the merging process, every non-zero pixel x in row r has associated with it a pair $(C, [a, b])$, meaning that pixel x belongs to component C with $R[C] = [a, b]$. Observe that component C induces on row r at least one contiguous interval, with each interval being formed by pixels belonging to component C. Every non-zero pixel in row $r+1$ has associated with it a pair $(C', [a', b'])$ with the analogous meaning. We merge components by performing the following three steps:

1. A left-to-right scan which processes the pairs in rows r and $r+1$ from left to right. At the end of the left-to-right scan, the processor in row r (resp. $r+1$) and column c contains the correct component and range information when only the entries in the first c columns are taken into account.

2. A right-to-left scan which processes the (updated) pairs in rows r and $r+1$ from right to left and generates the correct component and range information for rows r and $r+1$.

3. A broadcast to update the labels and associated ranges stored in the remaining processors of the two rectangular submeshes.

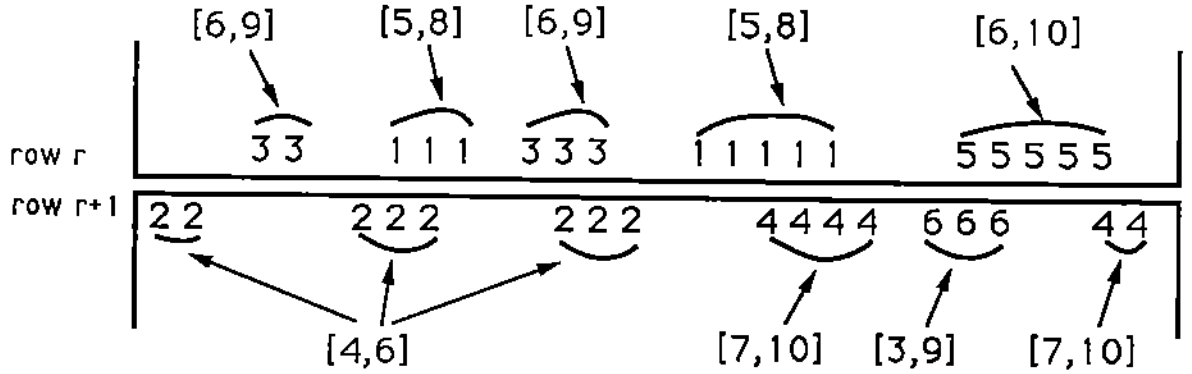
In order to ensure the correctness of the left-to-right scan, the timing is done as follows. Assume the left-to-right scan is initiated at processor $P_{r,0}$ at time t. Then, the left-to-right scan is active in processor $P_{r,c}$ at time $t + 2c$; i.e., it moves to the right every second time unit. When the left-to-right scan is active in processor $P_{r,c}$, it may initiate an update process which moves to the right ahead of the left-to-right scan; i.e., an update process moves one processor to the right at every time unit. This timing ensures that by the time the left-to-right scan reaches column c, all update processes initiated in columns $0, 1, \ldots, c-1$ have already been incorporated into the information stored in the pair at column c. An example is given in Figure 2 for $\epsilon = 6$. Figure 2(a) shows an initial labeling in rows r and $r+1$; Figure 2(b) shows the labels and ranges in these two rows after the left-to-right scan, and Figure 2(c) shows the labels after the right-to-left scan.

We now give a complete description of the merging process performed during the i-th iteration of our $\Theta(n \log n)$ time algorithm. The contents of processors $P_{r,c}$ and $P_{r+1,c}$ are as described above.

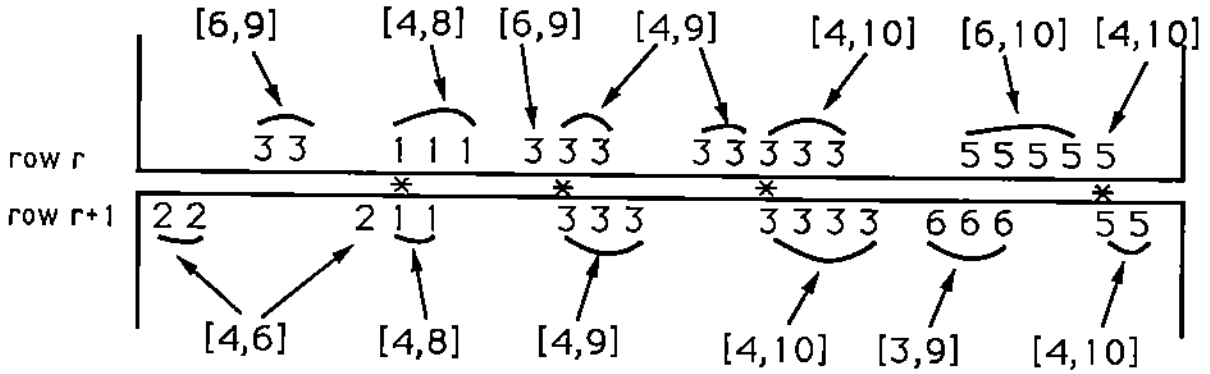
Algorithm 1: Merging process during the i-th iteration
Input: An $(i-1)$-restricted labeling, as well as the integer i, $1 \leq i \leq \log_2 n$.
Output: An i-restricted labeling.

(1) **Left-to-right scan**
Assume t is the time at which the left-to-right scan is initiated. Every processor $P_{r,c}$ with $r = 2^i j - 2^{i-1} - 1$, $1 \leq j \leq n/2^i$, contains the pair $(C, [a, b])$, meaning that pixel $D(r, c)$ belongs to component C with $R[C] = [a, b]$. Analogously, every processor $P_{r+1,c}$ with $r = 2^i j - 2^{i-1} - 1$, $1 \leq j \leq n/2^i$, contains the pair $(C', [a', b'])$. At time $t' = t + 2c$, processors $P_{r,c}$ and $P_{r+1,c}$ perform the following.

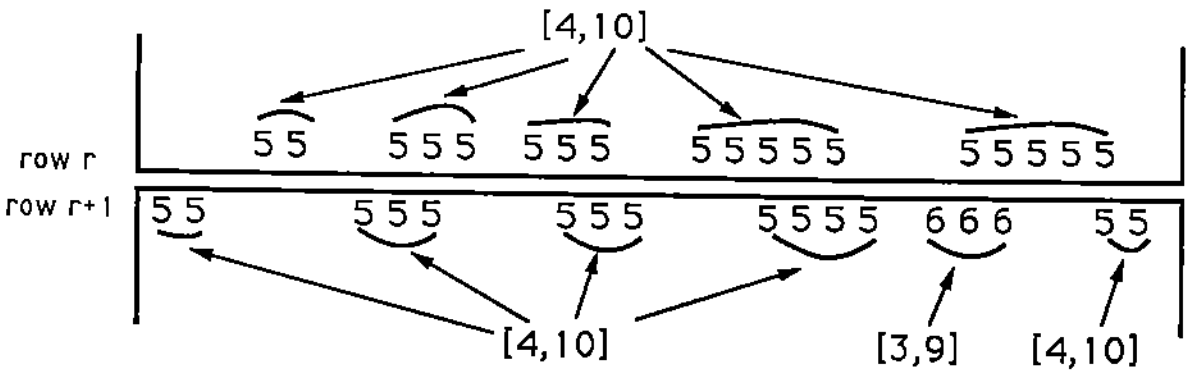
- If $C = C'$, then the initially disjoint components have already been merged by update processes initiated in columns to the left of c. No action is taken.
- If $C \neq C'$ and components C and C' cannot be merged to form a new valid component, no action is taken.
- If $C \neq C'$ and components C and C' can be merged to form a new valid component, the following actions are taken.
  * Select C as the label of the new component and determine the (possibly) new range of C. Processors $P_{r,c}$ and $P_{r+1,c}$ now each store the pair $(C, [\min(a, a'), \max(b, b')])$. Furthermore, they create the *update record* $(C, C', [\min(a, a'), \max(b, b')])$.



(a) Initial labeling and ranges of the components.



(b) Labeling and ranges after the left-to-right scan.
The *'s indicate the columns in which a merge was initiated.



(c) Labeling and ranges after the right-to-left scan.

Figure 2: Illustrating the merging process for a gray-scale image with $\epsilon = 6$; entries in rows r and $r + 1$ indicate component numbers of non-zero pixels.

* Processors $P_{r,c}$ and $P_{r+1,c}$ each record that a merge was initiated and that label C' has been changed to label C.
* Initiate a broadcast of the update records to the right, so that they are available to processors $P_{r,c+1}$ and $P_{r+1,c+1}$, $r = 2^i j - 2^{i-1} - 1$, $1 \leq j \leq n/2^i$, at time $t'+1$. At time $t'+1$, processors $P_{r,c+1}$ and $P_{r+1,c+1}$ use this information to update their pairs, as appropriate, and then continue passing the update records to the right, until they reach the rightmost processor in the row.

(2) **Right-to-left scan**

The right-to-left scan is initiated after the termination of the left-to-right scan, i.e., after the left-to-right scan has processed the information in column $n-1$. Every processor $P_{r,c}$ or $P_{r+1,c}$ that recorded the initiation of a merge, generates a *merge record* $(C, C''_{old}, [a,b])$, where C''_{old} represents the component label in processor $P_{r+1,c}$ just before the merge (i.e., at the beginning of time t'). The merge record is thus identical to the update record generated by that processor. The right-to-left scan consists of a right to left row rotation during which the merge records are passed to the left in lockstep fashion. So, every processor $P_{r,c}$ (and $P_{r+1,c}$) will see every merge record generated, in order; i.e., those merge records generated in $P_{r,c+1}$, $P_{r,c+2}$, $\ldots$, $P_{r,n-1}$ (resp., $P_{r+1,c+1}$, $P_{r+1,c+2}$, $\ldots$, $P_{r+1,n-1}$) as they "flow" through. A processor $P_{p,c}$, where $p = r+1$ or $p = r$, containing the pair $(C, [a,b])$ and receiving a merge record $(C_m, C'_m, [a_m, b_m])$ from processor $P_{p,c+1}$ executes the following before passing the merge record on to processor $P_{p,c-1}$.

- If $C = C_m$, then update $(C, [a,b]) \leftarrow (C, [a_m, b_m])$.
- If $C = C'_m$, then update the $(C, [a,b]) \leftarrow (C_m, [a_m, b_m])$.

(3) **Broadcast**

Broadcast the triples $(C_{orig}, C, [a,b])$, where C_{orig} represents the original label in a processor at the beginning of iteration i, and where C and $[a,b]$ represent the label and the range at termination of iteration i, respectively, to the appropriate rectangular subimage. Every processor containing a pixel currently belonging to C_{orig} obtains the triple $(C_{orig}, C, [a,b])$ and performs the necessary updates to its record.

(4) End (Algorithm 1)

It is easy to see that both scans (i.e., steps (1) and (2)) are completed in $\Theta(n)$ time. A $\Theta(n)$ time random access read can be used to perform the broadcasting step. Hence, the i-th iteration of the second phase runs in $\Theta(n)$ time and the overall running time of the algorithm is as claimed.

Theorem 1 *Given an $n \times n$ digitized gray-scale image stored one pixel per processor in a mesh of size n^2, the gray-scale component labeling problem can be solved in $\Theta(n \log n)$ time.* □

We next describe how the algorithm can be modified to run in $\Theta(n)$ time. We still use a divide-and-conquer strategy. We initialize every pixel to be a component by itself with its gray-scale value being its range. Then, instead of merging adjacent rows, we merge subsquares. The algorithm proceeds in stages, where at each stage disjoint subimages are merged into subimages twice their size. At the k-th stage disjoint subimages of size $2^{k-1} \times 2^{k-1}$ are merged into subimages of size $2^k \times 2^k$, as follows. Perform a horizontal merge between the northern $2^{k-1} \times 2^{k-1}$ western quadrant, and its neighboring $2^{k-1} \times 2^{k-1}$ eastern quadrant. Simultaneously, perform a horizontal merge between the southern pairs of quadrants. The result of this horizontal merge are two labeled $2^k \times 2^{k-1}$ subimages. The horizontal merge is followed by a single vertical merge, which merges the two $2^k \times 2^{k-1}$ subimages just generated into a subimage of size $2^k \times 2^k$. The merges use the merging scheme described for the $\Theta(n \log n)$ time algorithm and are thus completed in $\Theta(2^k)$ time. Therefore, the running time of stage k is $\Theta(2^k)$, and the running time of the algorithm obeys the recurrence $T(n^2) = T(n^2/4) + \Theta(n)$, which is $\Theta(n)$.

Theorem 2 *Given an $n \times n$ digitized gray-scale image store one pixel per processor in a mesh of size n^2, the gray-scale connected component problem can be solved in optimal $\Theta(n)$ time.* □

In both the $\Theta(n \log n)$ and the $\Theta(n)$ time algorithms, two δ-adjacent pixels with the same gray-scale value can be assigned to different valid components. (Notice that the $\Theta(n \log n)$ time algorithm can only assign adjacent pixels with the same gray-scale value to different components when the pixels are in the same column.) In the next two sections, we describe a graph-based algorithm for solving the gray-scale component labeling problem that avoids this property.

3 Algorithms for Finding Independent Sets

Our second algorithm for solving the gray-scale component labeling problem relies on an algorithm for finding a large independent set of a planar graph. In this section, we give the necessary graph theoretical terms and definitions, as well as our large independent set algorithm.

Let $G = (V, E)$ be an undirected graph. Let N and M denote the number of vertices and the number of edges of G, respectively. The subgraph of G *induced by* a subset $S \subseteq V$

is the graph consisting of the vertices in S and the edges of G that have both end vertices in S. If $(v,u) \in E$, we say u is a *neighbor* of v. $N(v)$ denotes the set of neighbors of v and $deg(v) = |N(v)|$ denotes the number of neighbors of v. For a subset $S \subseteq V$, we use $N(S)$ to denote the set of the vertices that are not in S, but are neighbors of vertices in S; i.e., $N(S) = (\cup_{v \in S} N(v)) - S$.

An *independent set* of G is a subset $I \subseteq V$ such that no two vertices in I are adjacent to each other. A *maximal independent set* (MIS) of G is an independent set I of G such that there exists no independent set of G that properly includes I. A *k-coloring* of G is an assignment of a color to each vertex of G such that (*i*) no two adjacent vertices are assigned the same color, and (*ii*) a total of k colors are used. Notice that if a k-coloring of G is given, then the set of the vertices with the same color is an independent set of G. Hence, a k-coloring of G is equivalent to a partition of V into k disjoint independent sets.

Let $\Delta = \Delta(G)$ denote the maximum degree of G. If $\Delta \leq c$, for a fixed constant c independent of N, then G is a *bounded degree* graph. First, we present an algorithm for constructing a $\Delta+1$ coloring of a bounded degree graph G. Then, we describe an algorithm for finding a maximal independent set of G using a $\Delta+1$ coloring of G. This algorithm is the building block of the large independent algorithm for planar graphs which we present at the end of this section. Similar ideas were used in the PRAM algorithms for solving these problems [6].

The basic idea of the $\Delta+1$ coloring algorithm is as follows. First, we partition the edge set E of $G = (V,E)$ into K (for some $K \leq \Delta$) subsets $E_1, \ldots, E_K$ such that for each i $(1 \leq i \leq K)$, the subgraph of G induced by E_i is a forest. We start the coloring process by considering the graph with vertex set V and empty edge set. Since this graph has no edges, all vertices can be colored by using the same color. The algorithm then performs a K-stage loop. At stage i $(i = K, \ldots, 1)$, we bring the edge set E_i back into the graph. The colors of the vertices are adjusted so that we still have a valid coloring of the current graph. The fact that the subgraph induced by E_i is a forest makes the color adjustment easy. After the K-th iteration, all edges of G have been brought back and we have generated a valid coloring of the original graph G. In the following algorithm, let $\{1, 2, \ldots, \Delta+1\}$ denote the set of the colors.

Algorithm 2: $\Delta+1$ coloring
Input: A bounded degree graph $G = (V,E)$ with maximum degree Δ.
Output: A $\Delta+1$ coloring of G.

(1) Set $i = 1$ and repeat:

Find a spanning forest F_i of G. Let E_i be the edge set of F_i.
$E \leftarrow E - E_i$; $i \leftarrow i+1$.

Until $E = \emptyset$.

$K \leftarrow i$.

Comment: Each time the loop is executed, the degree of any non-isolated vertex is reduced by at least 1. So, the loop is executed at most $K \leq \Delta$ times. After K executions, E becomes empty. Hence, the original set of edges has been partitioned into K subsets E_i $(1 \leq i \leq K)$ with each E_i being a forest.

(2) For all $v \in V$, color v by $C(v) = 1$.

Comment: Color each vertex $v \in V$ by the same color 1. Since E is empty now, this is a valid initial coloring.

(3) For $i = K$ down to 1 do:

(3.1) For each tree T in the forest F_i in parallel do:

Fix a root vertex r. Calculate the distance from each vertex v to r in T. If the distance is even, assign v a color 1. If the distance is odd, assign v a color 2. This gives a 2-coloring of F_i. Let $C'(v)$ denote the color of v obtained this way.

(3.2) $E \leftarrow E \cup E_i$;

(3.3) For $j = 1$ to $\Delta + 1$ do:

For all $v \in V$ such that $C(v) = j$ and $C'(v) = 2$, in parallel do:
$C(v) \leftarrow \max\{\{1, 2, \ldots, \Delta + 1\} - \{C(w) | (v, w) \in E\}\}$.

(4) End (Algorithm 2)

Comment on step (3.3): Before (3.3) is executed, each vertex $v \in V$ has a color consisting of two components $(C(v), C'(v))$. This represents a valid coloring of the current graph G using $2(\Delta + 1)$ colors. Step (3.3) finds a $\Delta + 1$ coloring for the current G by eliminating the colors $\{(j, 2) | 1 \leq j \leq \Delta + 1\}$. For each vertex v colored by $(j, 2)$ $(1 \leq j \leq \Delta + 1)$, v is recolored with a color from $\{1, \ldots, \Delta + 1\}$ that is not used by its neighbors in the current G. Since v has at most Δ neighbors in the current G, we can always find a free color for v. Since the vertices with the same color are independent, they can be recolored in parallel.

After a $\Delta + 1$ coloring of a bounded degree graph G is found, the following algorithm finds an MIS for G.

Algorithm 3: MIS for bounded degree graphs
Input: A graph $G = (V, E)$ with maximum degree Δ, and a $\Delta + 1$ coloring of G.
Output: An MIS I of G.

(1) $I \leftarrow \emptyset$.

(2) For $i = 1$ to $\Delta + 1$ do:

$V' \leftarrow \{$ the set of vertices in the current V with color $i\}$.
$I \leftarrow I \cup V'$; $V \leftarrow V - (V' \cup N(V'))$.

(3) Output I.

(4) End (Algorithm 3)

Observe that, if $G = (V, E)$ is a graph with maximum degree Δ, then any MIS of G contains at least $|V|/(\Delta + 1)$ vertices. For bounded degree graphs, Δ is independent of $N = |V|$, and thus an MIS contains at least a fixed fraction of the vertices in G.

We now discuss the implementation of these algorithms on a mesh. We assume the most general form of graph input for the mesh; namely that the edges of G are distributed arbitrarily with one edge per processor. Associative read and write operations that simulate the concurrent read and write capabilities of a PRAM can be performed in $\Theta(M^{1/2})$ time on a mesh of size M [11]. These algorithms are based on optimal mesh sorting algorithms which also require $\Theta(M^{1/2})$ time in the worst case. Finally, a number of tree-based algorithms can also be performed in optimal $\Theta(M^{1/2})$ time on the mesh [2, 19]. These algorithms include determining a spanning forest, orienting the spanning forest, computing generalized ancestor and descendant functions.

Since Δ is a constant, each step of Algorithm 2 and Algorithm 3 has an $O(M^{1/2})$ running time. This gives the following.

Theorem 3 *Given the M edges of a bounded degree graph G arbitrarily distributed one per processor on a mesh of size M, an MIS of G can be determined in $\Theta(M^{1/2})$ time.* □

We now turn to the algorithm for finding a large independent set of a planar graph. Let $G = (V, E)$ be a planar graph with N vertices and M edges. Since G is planar, we have $M \leq 3N$ [3]. We say an independent set I of G is a *c-large independent set* if

- $|I| \geq cN$, for some constant $c > 0$, and
- for any $v \in I$, $deg(v) \leq 6$.

A critical step of our second gray-scale component labeling algorithm is finding a c-large independent set of a planar graph $G = (V, E)$ stored in the mesh with one edge per processor. This can be done as follows. Let $V_1 = \{v \in V \mid deg(v) \leq 6\}$, and let $G_1 = (V_1, E_1)$ be the subgraph of G induced by V_1. Let n_i $(1 \leq i \leq d)$ be the number of vertices in

V with degree i, where d is the maximum degree of G. Then, $\sum_{i=1}^{6} n_i + 7\sum_{i=7}^{d} n_i \leq \sum_{i=1}^{d} i\, n_i = \sum_{v \in V} deg(v) = 2|E| \leq 6N$. Since $|V_1| = \sum_{i=1}^{6} n_i$ and $\sum_{i=7}^{d} n_i = N - |V_1|$, we have $|V_1| + 7(N - |V_1|) \leq 6N$. This implies that $|V_1| \geq N/6$.

Since the maximum degree of G_1 is 6, we can find a maximal independent set, call it I, of G_1 by using Algorithm 3. Clearly, I is an independent set of G and each vertex $v \in I$ has degree at most 6. Since $|V_1| \geq N/6$ and $|I| \geq |V_1|/7$, we have $|I| \geq N/42$.

Determining those vertices with degree less than or equal to 6 can be done by a constant number of sorting and prefix operations. Applying Algorithm 3 to the resulting subgraph allows us to state the following result.

Theorem 4 *Given the M edges of a planar graph G arbitrarily distributed one edge per processor on a mesh of size M, a $1/42$-large independent set of G can be determined in optimal $\Theta(M^{1/2})$ time.* □

4 Graph-Based Gray-Scale Component Labeling Algorithm

In this section, we present our second mesh algorithm for labeling a gray-scale digitized picture. Two major differences between this algorithm and the divide-and-conquer algorithm presented in Section 2 are that in this algorithm *(i)* the components are labeled by working with a graph representation of the image and not the image itself, and *(ii)* new components are formed by taking into account properties and characteristics of the components other than the validity of the resulting new range. First, we give an overview of the algorithm.

The first step of the algorithm generates a planar graph $G_0 = (V_0, E_0)$. Each vertex in V_0 represents a non-zero (i.e., a non-background) pixel of the gray-scale digitized image D. Two vertices x and y are defined to be adjacent in G_0 if and only if the pixels that they represent have the same gray-scale value and are δ-adjacent in D. We next compute the connected components of graph G_0 in $\Theta(n)$ time. This can be done by using either the algorithm in [17], which assumes that the edges of an undirected graph are arbitrarily distributed with one edge per mesh processor, or by a modification of the image-based component labeling algorithm given in [12].

The connected components are determined over a sequence of $O(\log n)$ stages, with each stage working with a planar graph and using a $1/42$-large independent set of this graph to guide decisions on which components to merge. Let $G_i = (V_i, E_i)$ be the graph at the beginning of the ith stage, $i \geq 1$. At any time a vertex v of G_i represents a valid component C_v of D with range $R(v) = [a_v, b_v]$. (Recall that a_v is the minimum gray-scale value of any pixel in C_v, b_v is the maximum gray-scale value of any pixel in C_v, and a component is valid if and only if $0 \leq b_v - a_v \leq \epsilon$.) We say that such a vertex is *valid*. For two vertices

$u, v \in V_i$, u and v are adjacent in E_i if and only if there exists a pixel x in C_u and a pixel y in C_v such that x and y are δ-adjacent in the digitized picture D.

The stage 1 graph $G_1 = (V_1, E_1)$ is defined as follows. The vertices of G_1 correspond to the connected components of G_0. For every vertex v of G_1, the range of v is $R(v) = [a, a]$, where a is the gray-scale value of the pixels belonging to C_v. G_1 is clearly a planar graph, and we have $|V_1| \leq n^2$ and $|E_1| \leq 3|V_1| = O(n^2)$. (Notice that if one wanted to force a set of connected pixels to belong to the same component, even if the set is not valid, this information could be incorporated into the formation of G_1.)

Consider stage i and the associated graph $G_i = (V_i, E_i)$, $i \geq 1$. Let u and v be two vertices of G_i. We say u and v are *compatible* if and only if $(u, v) \in E_i$ and $\max\{b_u, b_v\} - \min\{a_u, a_v\} \leq \epsilon$. If vertices u and v are compatible, the valid components C_u and C_v associated with them can be merged to form a new (larger) valid component. A vertex $v \in V_i$ is called *maximal* if v is not compatible with any of its neighbors in G_i. Note that v is a maximal vertex if and only if the gray-scale connected component C_v is a maximal component.

The first step of stage i is to find a 1/42-large independent set I of G_i using the algorithm associated with Theorem 4. During the processing of stage i, one of the following will occur for each vertex $v \in I$.

1. v will be merged with a compatible neighbor.

2. It will be determined that v is a maximal vertex (i.e., C_v is a maximal valid component), and v will be deleted from G_i.

Therefore, at the conclusion of stage i, all vertices in I will have been removed from G_i. Since $|I| \geq \frac{1}{42}|V_i|$, we have reduced the number of vertices in G_i by a fixed fraction. Hence, in $O(\log n)$ stages, the graph under consideration becomes empty and a set of maximal valid components is found.

During stage i, there are two steps in which we have a certain amount of freedom in how new components are formed from existing ones. Before giving a more complete description of the algorithm, we describe different criteria and heuristic approaches for selecting components to be merged. Let v be a vertex of G_i representing component C_v with range $[a_v, b_v]$. Assume v is adjacent to vertices $v_1, v_2, \ldots, v_k$ and merging component C_v with any component C_{v_j}, $1 \leq j \leq k$, results in a valid component. During stage i, each vertex in independent set I selects one of its neighboring vertices. This selection may, in some later step of stage i, lead to a merge of the components associated with the two vertices. Let v_s be the vertex selected by a vertex v in I. Selecting v_s arbitrarily obviously results in a valid labeling. However, by maintaining additional information about the components, it is possible to use some insight in selecting which components to merge. Notice that in our

model of computation, any additional information about a component must be stored in a constant number of registers. Further, it must be possible to update such information in constant time when components are merged. Examples of additional information that can be stored about components include the following.

- The number of pixels in a component.
  This can easily be done by associating with every vertex v a quantity $nr\_p(v)$ representing the number of pixels in component C_v.

- The average gray-scale value of the pixels in a component.
  Let $av(v)$ be this average gray-scale value. If we also maintain $nr\_p(v)$, this average can be maintained and updated in $O(1)$ space and time, respectively.

- The smallest rectangle enclosing all the pixels in a component.
  For any component C_v, this rectangle can be represented by the coordinates of its lower left corner and its upper right corner, respectively, thus requiring a total of 4 registers. When two components are merged, the new coordinates are determined in $O(1)$ time.

Therefore, possibilities for selecting component C_{v_s} include the following.

1. Minimize the change of range.
   Select vertex v_s so that $|\max(b_v, b_{v_s}) - \min(a_v, a_{v_s})|$ is a minimum over all vertices v_j, $1 \leq j \leq k$.

2. Maximize the number of pixels.
   Select vertex v_s so that $nr\_p(v_s)$ is a maximum over all other vertices.

3. Minimize the change in the average gray-scale value.
   Select vertex v_s so that $|\frac{av(v) \times nr\_p(v) + av(v_s) \times nr\_p(v_s)}{nr\_p(v) + nr\_p(v_s)} - av(v)|$ is a minimum over all other vertices.

4. Maximize the overlap between smallest enclosing rectangles.
   Select vertex v_s so that the overlap between the rectangle associated with component C_{v_s} and the rectangle associated with component C_v is of maximum size. Observe that the rectangles of two adjacent vertices can be disjoint. In such a case the rectangles are adjacent and the amount of adjacency can be used as the quantity to be maximized.

Since the main focus of this paper is on the design of efficient mesh algorithms for the gray-scale labeling problem, we do not discuss the situations under which and for what

images each one of the above selection criteria is most appropriate. In fact, a combination of selection criteria may actually be desirable. For example, it may be desirable to use the criteria for minimizing the change in range, though when many vertices result in a very similar range change, one might use another criteria in order to distinguish which merger might be most desirable. When in our algorithm a vertex v needs to select a vertex v_s among its neighbors, we use the term "v selects v_s using specified selection rules," implying that appropriate selection rules have been specified.

During stage i, a vertex v may be required to make a somewhat different selection among its neighbors $v_1, v_2, \ldots, v_k$. Assume, as above, that merging component C_v and with any component C_{v_i} results in a valid component. We now need to determine a set T', $T' \subseteq \{v_1, v_2, \ldots v_k\}$, such that merging component C_v and the components associated with the vertices in T' results in a maximally valid component. Notice that selection criteria 1-4 can be used to guide the selection of vertices for T'. However, using some of the criteria alone (e.g., minimizing the change of the range) may not satisfy the condition of set T' being maximal. Possible examples are using the maximum number of vertices, where, among all subsets T' containing the same number of vertices, another selection criteria is used to break ties. Or, to select the vertices resulting in a component consisting of the maximum number of pixels, again using another criteria to break ties.

We now return to the description of the algorithm and give the details of stage i. At any stage i, $G = (V, E)$ denotes graph $G_i = (V_i, E_i)$.

Algorithm 4: Gray-scale component labeling
Input: A gray-scale digitized picture D.
Output: A set P representing maximal valid components of D.

(1) Construct the planar graph G_1, as described. Denote it by $G = (V, E)$.

(2) $P = \emptyset$.

(3) While V is not empty Do:

(3.1) Find a 1/42-large independent set I of G.

(3.2) Do the following statements 6 times:

(a) For every $v \in I$, examine the (at most 6) neighbors of v in V.

* If v is not compatible with any of its neighbors, then v is a maximal vertex and it corresponds to a maximal valid component C_v. In this case, delete v from V, remove v from I, and add C_v, the gray-scale connected component corresponding to v, to P.

* If v has at least one compatible neighbor, then v selects among its compatible neighbors vertex v_s using specified selection rules. Let $F = \{(v, v_s) \mid v \in I$ and v_s is the selected compatible neighbor of $v\}$.

(b) Let H be the subgraph of G induced by the edge set F. Since I is an independent set of G, every connected component of H is a "star" graph. (A star graph consists of a *center* vertex not in I and several *tip* vertices. Each tip vertex is adjacent to the center vertex.)

(c) For every star S of H, perform the following operations. Let x be the center vertex of S and let $T = \{t_1, \ldots, t_k\}$ be the set of the tip vertices of S. Find a maximal subset $T' \subseteq T$ using specified selection rules. T' satisfies the following: (i) Merging all components associated with the vertices in T' with component C_x forms a new valid component and (ii) including the component associated with any vertex not in T' violates the validity condition. The merge of the vertices in T' into x is done by removing the vertices in T' from V, updating $R(x)$ to reflect the new range, and removing the vertices in T' from I.

(4) End (Algorithm 4)

After each iteration of the Do loop (step 3.2), for every vertex $v \in I$, either

1. v is a maximal vertex and is deleted from G, or
2. v is merged with a compatible neighbor, or
3. the number of compatible neighbors of v is reduced by at least one.

Since every $v \in I$ has at most 6 neighbors, after the 6-th iteration of the Do loop, v is either deleted from G or merged with a compatible neighbor.

Therefore, each iteration of the While loop (step 3) will remove all vertices in I and hence reduce the size of G by a factor of $41/42$. Thus after at most $\log_{42/41} n^2$ iterations of the loop, G becomes empty and a maximal valid partition P is known.

We now discuss some of the implementation details of this algorithm. During stage i, every (undirected) edge of G which connects vertex u with vertex v will be represented twice, once with u as the key, denoted as (u, v), and once with v as the key, denoted as (v, u). Notice that this may require a mesh of size M to emulate a mesh of size $2M$, which is a standard technique [11] when dealing with data sets that are a constant size larger than the number of available processors. Using such a virtual machine effects the constants of proportionality, but not the asymptotic running times of the algorithms. These

(keyed) edges are stored one edge per processor in the smallest possible square submesh containing processor $P_{0,0}$. Suppose processor p is responsible for the (keyed) edge (u, v). Then processor p will contain

- vertex u in the key field, with other fields that include
- vertex v,
- the range $[a_{C_v}, b_{C_v}]$ of C_v,
- the range $[a_{C_u}, b_{C_u}]$ of C_u, and
- other entries needed during the component selection process.

Step (3.1) is accomplished by using the algorithm described in Theorem 4 to find a 1/42-large independent set I of G_i. The details of steps 3.2(a) and (b) are straightforward and are omitted. We describe two solutions for step 3.2(c). The first one determines a set T' of maximum cardinality that, among all sets of maximum cardinality, minimizes the change in the new range (with respect to the original range of vertex x). Its $O(n)$ running time makes use of the assumption that the gray-scale value of a pixel is bounded by g with $g \leq c \log n$. The second solution determines a maximal set and its $O(n)$ running time holds even for unbounded gray-scale values.

Our solution for determining a set T' of maximum cardinality first sorts the edges of every star S of H into snake-like order according to the left endpoint of the range (i.e., the a_{t_j}'s) associated with every edge (t_j, x), $1 \leq j \leq k$.

- For edges $(t_j, x), (t_{j+1}, x), \ldots, (t_{j+l}, x)$ of S with identical left and right endpoints (i.e., $a_{t_p} = a_{t_{p+1}}$ and $b_{t_p} = b_{t_{p+1}}$ with $j \leq p < j + l$) we record only one entry, together with a frequency count indicating how many components have this range. After duplicate ranges are removed, we compress the remaining edges of S into contiguous locations.
- Every edge with $a_{t_j} \leq a_x$ initiates a scan to determine the maximum number of components that can be merged with component C_x when a_{t_j} is the left endpoint of the new component to be formed. The right endpoint of this component can be at most $a_{t_j} + \epsilon$. Should no entry with $a_{t_j} = a_x$ exist, we further initiate a scan with a_x as the left endpoint of the new component. The scan initiated by edge (t_j, x) visits all edges with larger left endpoints in order. If the scan is at entry (t_{j+l}, x) and $b_{j+l} \leq a_{t_j} + \epsilon$, then the edges having range $[a_{j+l}, b_{j+l}]$ are included and the right endpoint of the component being formed is updated accordingly. Entries with $b_{j+l} > a_{t_j} + \epsilon$ are skipped. The scan terminates when an edge with a left endpoint larger than $a_{t_j} + \epsilon$ is encountered. During each scan we thus determine the maximum

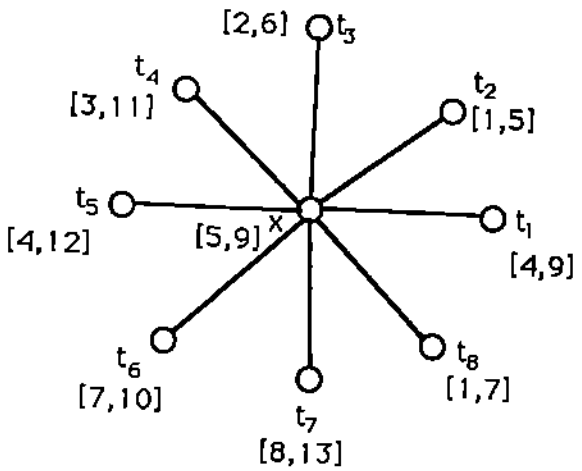


Figure 3: A star graph with associated ranges and $\epsilon = 9$.

number of components that can be included and the associated range of the new component whose left endpoint would be a_{t_j}.

- Select the left endpoint resulting in a set T' of maximum cardinality and minimizing the range change for component C_x.

For the star graph shown in Figure 3 the scan initiated at edge (t_2, x) with 1 as the left range and it includes altogether five edges, namely, $(t_2, x), (t_8, x), (t_3, x)$, (t_1, x), and (t_6, x). The resulting new component has a range of [1,10] and it maximizes the number of components that can be merged with C_x. The $O(n)$ time bound is shown as follows. During a scan, either the right or the left endpoint of the range associated with an entry increases by at least one. Thus, a scan visits at most ϵ^2 entries. Since, by assumption, $\epsilon = O(\log n)$, each scan terminates in $O(\log^2 n)$ time. All other steps (i.e., sorting, compressing, selecting the maximum) can easily be completed in $O(n)$ time.

Our second solution for finding a set T' does not make any assumption about the gray-scale values. It generates a set T' that is maximal and it uses specified selection rules to break ties. The first step is again a sort: the edges of every star S of H are sorted into snake-like order by the vertices *not* in the independent set I. For any star graph S with center x, the edges $(x, t_1), \ldots, (x, t_k)$ are then in contiguous locations.

- Find an element l of S maximizing $b_l - a_l$ with $a_l \leq a_x$ and $b_l \geq b_x$, breaking ties according to specified selection rules. If there is no such l, then let $[a, b] = [a_x, b_x]$, else $[a, b] = [a_l, b_l]$. That is, we find the largest interval associated with a tip vertex that completely encloses the interval associated with the center vertex. Obviously, the new interval is valid. For the star graph shown in Figure 3, this step will choose $l = 4$ and $[a, b] = [3, 11]$.

- Find an element of S, call it l', with the smallest $a_{l'}$, such that $b - a_{l'} \leq \epsilon$ and $b_{l'} \leq b$, breaking ties as in the previous step. If no such l' exists, then leave $[a, b]$ as is, else set $[a, b] = [a_{l'}, b]$. This operation reduces the left end of the interval as much as possible, without exceeding the given ϵ value. For the star graph of Figure 3 this step will generate $l' = 3$ and $[a, b] = [2, 11]$.

- Find an element of S, call it l'', with the largest $b_{l''}$, such that $b_{l''} - a \leq \epsilon$ and $a_{l''} \geq a$, breaking ties as in the previous steps. If no such l'' exists, then leave $[a, b]$ as is, else set $[a, b] = [a, b_{l''}]$. This operation increases the right end of the interval as much as possible, again, without exceeding the given ϵ value. For the star graph of Figure 3 this third step cannot increase the right range any further. The generated range remains $[2, 11]$ and set T' contains four entries.

- All $t_i \in T$ with ranges in $[a, b]$ are in T'.

At the conclusion of stage i, we record that vertices of T' did get merged and we determine the new maximal vertices. The information needed for stage $i + 1$ is then compressed into a smaller square submesh. As already stated, the number of vertices, and hence the number of edges, is reduced by at least a factor of 41/42. This data compression is a well-known mesh technique for algorithms which continually eliminate a fixed fraction of the data. The reader may wish to refer to [11], and the references contained therein, for other examples of such algorithms. The information about which vertices are merged and which are maximal is kept in the submesh used during stage i, but outside the submesh for stage $i + 1$. This information is needed during the back-propagation step that is initiated after step (2) of the algorithm is completed. The task of the back-propagation is the actual assignment of labels to the pixels of picture D.

Every stage i can be performed on a mesh with a fixed number of fundamental data movement operations, such as sorting, random access read, broadcasting within ordered intervals, and parallel prefix within ordered intervals. As stated previously, given a mesh of size n^2, these operations can all be performed in $O(n)$ time. Therefore, the running time of the algorithm obeys the recurrence $T(n) \leq T(\frac{41}{42}n) + \Theta(n)$, which is $\Theta(n)$.

Theorem 5 *Given an $n \times n$ digitized gray-scale image stored one pixel per processor in a mesh of size n^2, the gray-scale component labeling problem can be solved in optimal $\Theta(n)$ time.* □

5 Conclusion

We have defined a form of connectivity for gray-scale images in which pixels belonging to the same component can have different gray-scale values and components satisfy a maxi-

mal property. Using this definition, we have presented two asymptotically optimal mesh algorithms for the component labeling problem. The first algorithm has advantages that include simplicity and the potential for a straightforward implementation. However, it has the drawback that neighboring pixels with the same label can be assigned distinct components. The second algorithm overcomes this problem and, in addition, allows that properties and characteristics of components are incorporated into the process of deciding which components to merge. This algorithm works with a graph representation of the image exploits a maximal independent set algorithm we develop for bounded degree graphs. Our MIS algorithm can also be used to solve the MIS problem and the 5-coloring problem for planar graphs. While these two problems are not related to the gray-scale component problem, their mesh solutions are of algorithmic interest. The MIS problem has emerged as a useful tool for solving other planar graph problems (e.g., 5-coloring, edge-coloring, straight-line embeddings) and the 5-coloring problem is a well-known problem in graph theory. We present these algorithms in the appendices.

Throughout we considered the problem of generating maximal labelings. Another version of the gray-scale labeling problem is to generate components so that each component is connected and valid and the total number of components is a minimum. It is not hard to generate gray-scale images for which our algorithms can produce solutions containing $O(n^2)$ connected components, whereas it is possible to label the pixels so that only $O(n)$ components are generated. As is the case for maximal labelings, partitioning into a minimum number of components does not necessarily generate unique labelings. An interesting open problem is that of proving whether or not the problem of determining the minimum number of connected and valid components is NP-hard.

Acknowledgment

We would like to thank Steve Tanimoto for suggesting the gray-scale component labeling problem.

References

[1] M.J. Atallah, F. Dehne, R. Miller, A. Rau-Chaplin, and J.-J. Tsay, Multisearch techniques for implementing data structures on a mesh-connected computer, *Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures*, 1991, pp. 204-214.

[2] M.J. Atallah and S.E. Hambrusch, Solving tree problems on a mesh-connected processor array, *Information and Control*, 69(103), 1986, pp. 168-187.

[3] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, North Holland, 1976.

[4] N. Chiba, T. Nashizeki, and N. Saito, A Linear 5-Coloring Algorithm of Planar Graphs, *J. Algorithms*, 2, 1981, pp. 28-35.

[5] F. Dehne and S.E. Hambrusch, Parallel Algorithms for Determining k-Width-Connectivity in Binary Images, *JPDC*, Vol. 12, Nr. 1, 1991, pp 12-23.

[6] A. V. Goldberg, S. A. Plotkin, and G.E. Shannon, Parallel Symmetry-Breaking in Sparse Graphs, *Proc. 19th ACM Symposium on Theory of Computing*, 1987, pp. 315-324.

[7] T. Hagerup, M. Chrobak, and K. Diks, Optimal Parallel 5-Coloring of Planar Graphs, *SIAM J. on Computing*, 18(2), 1989, pp. 288-300.

[8] X. He and R. Miller, Optimal Mesh Algorithms for Maximal Independent Subset and 5-Coloring, Tech. Rept. 90-27, Department of Computer Science, State University of New York at Buffalo, Oct., 1990.

[9] R.M. Karp and V. Ramachandran, Parallel algorithms for shared-memory machines, in *Handbook of Theoretical Computer Science*, Volume A, Algorithms and Complexity, MIT Press, 1990, pp. 869-942.

[10] F.T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes*, Morgan Kaufmann Publ., 1992.

[11] R. Miller and Q.F. Stout, *Parallel Algorithms for Regular Architectures*, The MIT Press, 1992, to appear.

[12] D. Nassimi and S. Sahni, Finding connected components and connected ones on a mesh-connected parallel computer, *SIAM Journal on Computing*, 9 (1980), 744-757.

[13] D. Nassimi and S. Sahni, Bitonic sort on a mesh-connected parallel computer, *IEEE Transactions on Computers*, C-27(1):2–7, January 1979.

[14] D. Nassimi and S. Sahni, Data broadcasting in SIMD computers, *IEEE Transactions on Computers*, C-30(2):101–107, February 1981.

[15] A. Montanvert, P. Meer, and A. Rosenfeld, Hierarchical image analysis using irregular tessellations, Tech. Rept. CAR-TR-464, University of Maryland, Sept. 1989.

[16] T. Pavlidis, *Algorithms for Graphics and Image Processing*, CSP, 1982.

[17] J. Reif and Q.F. Stout, Optimal component labeling algorithms for mesh computers and VLSI, to appear.

[18] A. Rosenfeld and A. Kak, *Digital Picture Processing* Academic Press, 1982.

[19] Q.F. Stout, Tree-based graph algorithms for some parallel computers, *Proc. of the 1985 International Conference on Parallel Processing*, 1985, pp. 727-730.

[20] C.D. Thompson and H.T. Kung, Sorting on a mesh-connected parallel computer, *Communications of the ACM*, 20(4):263–271, April 1977.

A MIS for Planar Graphs

In this section, we present an optimal mesh algorithm for constructing a maximal independent set of a planar graph $G = (V, E)$, $|V| = n$, $|E| = m$. The algorithm does not require a planar embedding as input. It is well known that $m \leq 3n$ for planar graphs [3]. This implies the average degree of vertices in G is at most 6.

Algorithm 5: MIS for Planar Graphs
Input: A planar graph $G = (V, E)$.
Output: An MIS of G.

(1) $I \leftarrow \emptyset$.

(2) Repeat:

(2.1) Let V' be the set of vertices of G with degree ≤ 6. Let G' be the subgraph of G induced by V'.

Comment: Since the average degree of the vertices in G is at most 6, we have shown in section 3 that $|V'| \geq |V|/6$. Also note that the maximum degree of G' is no more than 6.

(2.2) Find a maximal independent set I' in G' by using Algorithm 3.

(2.3) $I \leftarrow I \cup I'$.

(2.4) $V \leftarrow V - (V' \cup N(V'))$.

Until $V = \emptyset$.

(3) Output I.

(4) End (Algorithm 5)

Note that at step (2.4) all vertices in V' (and possibly some vertices not in V') are removed from V. Thus after each execution of the repeat loop, the size of the resulting graph is at most 5/6 of the size of the previous graph. Therefore, the repeat loop is executed $O(\log n)$ times.

Notice that if this algorithm were implemented in a straightforward fashion on a mesh of size m, the algorithm would run in $O(m^{1/2} \log n)$ time. In order to reduce the running time of the algorithm, a compression step is used at the end of every iteration. This step compresses the $O(k)$ pieces of information needed for the next iteration into the northwest submesh of size k. If the information is currently in a submesh of size K, then a $\Theta(K^{1/2})$ time sort-based operation can perform the compression [11]. Further, in order to avoid an

overaccumulation of data in any processor, all of the information not needed for the next iteration is moved so that it remains inside the current submesh of size K, but outside of the submesh of size k. Again, sort-based operations can perform this operation in $\Theta(K^{1/2})$ time.

Therefore, the running time of the algorithm is given by $T(m) \leq T(5m/6) + \Theta(m^{1/2})$, which is $\Theta(m^{1/2})$.

Theorem 6 *Given a planar graph $G = (V, E)$ such that the m edges of G are arbitrarily distributed one edge per processor on a mesh of size m, an MIS of G can be determined in optimal $\Theta(m^{1/2})$ time.* □

B 5-Coloring of Planar Graphs

The sequential and parallel complexity of 5-coloring planar graphs have been well studied. An $O(\log\log^* n)$ time $O(n/\log\log^* n)$ processor PRAM algorithm for 5-coloring planar graphs is given in [7] which is based on the linear time sequential algorithm in [4]. We show that this algorithm can be adapted to a mesh computer to yield an optimal mesh algorithm. It should be noted that although the graph must be planar, the algorithm does not require a planar embedding as input.

Let $G = (V, E)$, $|V| = n$, $|E| = m$, be a planar graph. Define a vertex $v \in V$ to be a *small vertex* if $deg(v) \leq 11$. Define a vertex $v \in V$ to be a *large vertex* if $deg(v) \geq 12$. A vertex $w \in V$ is called *reducible* if one of the following conditions holds:

(1) $deg(w) \leq 4$.

(2) $deg(w) = 5$, and w has at most one large vertex as its neighbor.

(3) $deg(w) = 6$, all neighbors of w are small vertices, and the subgraph induced by the neighbor set $N(w)$ contains a simple cycle $H(w)$.

For two non-adjacent vertices $u, v \in V$, the *identification* of u and v is an operation which replaces u, v and their incident edges by a new vertex z adjacent to exactly those vertices in $V - \{u, v\}$ that were adjacent to either u or v (or both) in the original graph. Identification of three pairwise non-adjacent vertices is defined analogously.

Definition: A *reduction* centered at a reducible vertex w is defined as follows:

(1) $deg(w) \leq 4$: delete w from G.

(2) $deg(w) = 5$: delete w from G, and identify two small non-adjacent vertices $x, y \in N(w)$.

(3) $deg(w) = 6$: delete w from G and either

(a) identify three pairwise non-adjacent vertices $x, y, z \in N(w)$, or

(b) identify two non-adjacent vertices $x_1, y_1 \in N(w)$ and identify two non-adjacent vertices $x_2, y_2 \in N(w)$ such that x_1, y_1, x_2, y_2 occur in the cyclic order on the simple cycle $H(w)$ of the subgraph induced by $N(w)$.

It is shown in [7] that for any reducible vertex w, a reduction centered at w exists. Let G' be the graph obtained from G by performing a reduction centered at a reducible vertex w. It is shown in [7] that G' is still planar. Therefore, a 5-coloring of G' can be found recursively. In order to extend the 5-coloring of G' to be a 5-coloring of G, we first undo the identification of the neighbors of w, letting each new vertex created in the process inherit the color of the vertex from which it is derived (note that vertices that are identified are always non-adjacent). Then put w back in place. By the coloring convention just mentioned, the neighbors of w are colored by at most 4 colors. Hence the fifth color can be used to color w.

In order to find a 5-coloring of G efficiently in parallel, we need to find a large set of reducible vertices so that the reductions performed on these vertices can be done in parallel.

Let A be the set of reducible vertices of G. It is shown in [7] that $|A| \geq n/196$. Let w_1 and w_2 be two vertices in A. If either w_1 and w_2 are adjacent in G, or w_1 and w_2 have a small vertex as their common neighbor, then the reductions centered at w_1 and w_2 might interfere with each other and hence cannot be done in parallel. Therefore, we first find a subset $I \subset A$ so that the reductions centered at the vertices in I can be performed in parallel. This is done by choosing an independent set $I \subset A$ in an auxiliary graph G_4 obtained from G. The algorithm is as follows.

Algorithm 6: 5-coloring of planar graphs
Input: A planar graph $G = (V, E)$.
Output: A 5-coloring of G.

(1) Find the set A of reducible vertices of G.

(2) Find the set $S = \{v \in V | deg(v) \leq 11\}$ of small vertices. Construct $G_1 = (S, E_1)$, the subgraph of G induced by S.

(3) Construct $G_2 = (S, E_2)$, the "square" graph of G_1. Namely, (x, y) is an edge in E_2 if and only if there is a vertex $z \in S$ so that both (x, z) and (z, y) are edges in E_1.

(4) Construct $G_3 = (V, E \cup E_2)$.

(5) Construct G_4, the subgraph of G_3 induced by the set A of reducible vertices.

Comment: the maximum degree of G_4 is bounded by a constant.

(6) Find a maximal independent set I in G_4 by using Algorithm 3.

Comment: Since G_4 is a bounded degree graph, I contains at least a fixed fraction of the vertices in G_4 (by the remark following the Algorithm 3). Since the vertex set A of G_4 satisfies $|A| \geq n/196$, $|I| \geq cn$ for some fixed constant $0 < c < 1$.

(7) Perform reductions for the vertices in I. Let G' be the resulting graph.

Comment: the size of G' is at most $c'n$ for a fixed constant $c' = 1 - c < 1$ as mentioned in the comment for Step (6). Also note that since a reducible vertex has degree at most 6, and the vertices to be identified have degree at most 11, the reduction centered at a reducible vertex can be done in constant time.

(8) Recursively find a 5-coloring of G'

(9) Undo the reductions performed in step (7), and color the vertices in I.

(10) End (Algorithm 6)

On a mesh, finding a reducible set of vertices is accomplished by a sort-based operation to group together all edges associated with each vertex. Finding the set S of small vertices is done by a prefix operation within ordered intervals (edges corresponding to each vertex) to tally the number of edges per vertex. Squaring a bounded degree graph can be accomplished by an associative read and interval broadcast operation so that each vertex determines all vertices that can be reached with exactly 2 edges. Since the size of G' is at most a fixed fraction of G, the recursion depth of the algorithm is $O(\log n)$. As with the Algorithm 5, care must be taken to compress the data for the start of each recursive step into a submesh of the appropriate size (leaving unnecessary data outside of this new submesh). Since sort-based operations and interval operations can be done in $\Theta(m^{1/2})$ time on a mesh of size m, the running time of the algorithm is given by the recurrence $T(m) \leq T(cm) + \Theta(m^{1/2})$, $c < 1$ a constant, which is $\Theta(m^{1/2})$.

Theorem 7 *Given a planar graph $G = (V, E)$ distributed arbitrarily one edge per processor on a mesh of size m, a 5-coloring of G can be determined in optimal $\Theta(m^{1/2})$ time.* □