International Journal of Foundations of Computer Science

Low-Latency Connected Component Labeling Using an FPGA

YASUAKI ITO and KOJI NAKANO
Graduate School of Engineering, Hiroshima University
1-4-1 Kagamiyama, Higashihiroshima, Hiroshima, 739–8527, Japan

Received (7 September 2008)
Revised (3 March 2009)
Communicated by Jacir L. Bordim

ABSTRACT

Connected component labeling is a process that assigns unique labels to the connected components of a binary image. The main contribution of this paper is to present a low-latency hardware connected component labeling algorithm for k-concave binary images designed and implemented in FPGA. Pixels of a binary image are given to the FPGA in raster order, and the resulting labels are also output in the same order. The advantage of our labeling algorithm is low latency and to use a small internal storage of the FPGA. We have implemented our hardware labeling algorithm in an Altera Stratix Family FPGA, and evaluated the performance. The implementation result shows that for a 10-concave binary image of 2048×2048, our connected component labeling algorithm runs in approximately 70ms and its latency is approximately 750μs.

Keywords: Connected component labeling, FPGA, Image processing, Hardware algorithm

1. Introduction

Connected component labeling is a process that assigns unique labels to the connected components of a binary (black and white) image as labels. As illustrated in Figure 1, each connected component of black pixels is assigned an integer value as a unique label. More specifically, each pixel assigned a label such that pixels in the same connected component have the same label and those in different connected components have different labels. In this figure, 3 connected components of black pixels have assigned labels 1, 2, or 3. There are two common ways of defining connectedness in a binary image, i.e., 4-connectedness and 8-connectedness [1]. In this paper, we use the 4-connectedness.

In the field of the image recognition, connected component labeling is a basic and important task as a preliminary step that finds the feature of the object [2, 3]. After the connected component labeling, connected components corresponding to different objects can be studied and recognized by higher level image analysis processes.

Since the component labeling is a basic and important task, there are a lot of published works. They are based on the algorithmic techniques as follows:

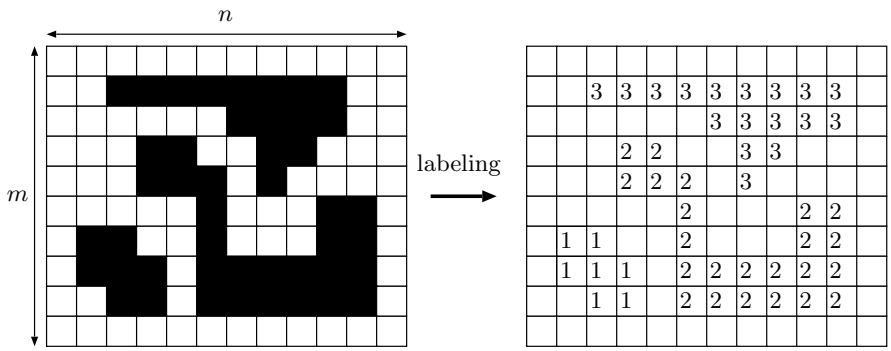


Figure 1: An example of connected component labeling for a binary image.

- *Temporary labeling method* [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]: Component labeling is done by two or more separated stages that scan the binary image in raster order. The first stage scans the input binary image and assigns each connected component temporary labels. No two connected components are assigned the same temporary label, but two or more temporary labels may be given to a connected component. The second stage scans the input binary image again, and chooses one of the temporary labels for each connected component. Sometimes, more than two stages are executed until each connected component has a unique label.

- *Contour labeling method* [17, 18]: This labeling method has two stages. The first stage traces the contour of each connected components and assigned labels to pixels in the contour. In the second stage, interior pixels are scanned and assigned labels.

- *Parallel processing approach* [19, 20, 21, 22, 23, 24, 25, 26, 27, 28]: Parallel labeling is performed with parallel machine models such as the use of mesh connected parallel processor arrays to speed-up the computation.

- *Hardware algorithm* [14, 26, 29]: Hardware implementations of the above algorithms have also been proposed in order to accelerate processing speed.

The main contribution of this paper is to present a hardware algorithm that performs low-latency connected component labeling using FPGAs. An *FPGA* (Field Programmable Gate Array) is a programmable VLSI in which any circuit can be embedded quickly. Typical FPGAs consist of an array of programmable logic elements, distributed memory blocks, and programmable interconnections between them. The logic block usually contains either a four-input logic function or a multiplexer and several flip-flops. The distributed memory block is usually a dual-port RAM on which a word of data for possibly distinct addresses can be read/written at the same time.

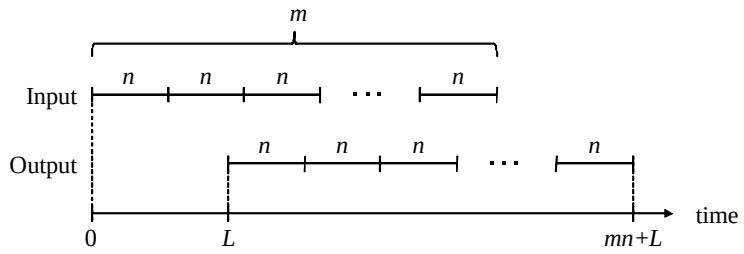


Figure 2: Process of our labeling hardware for an input image of $m \times n$.

It is difficult to implement the temporary labeling method for connected component labeling in the FPGAs. The temporary labeling method needs to store temporary labels for all pixels. However, the total size of memory blocks of the FPGAs is no more than few megabits. If the size of image is 2048×2048 and temporary labels have 16 bits, at least 64 megabits storage is necessary, which is too large for current FPGAs. Of course, if a large external storage such as DRAMs is available, all temporary labels of whole image can be stored. However, the use of a large external storage is costly in terms of price and power consumption. The use of external storage may not be possible if FPGAs are used for small and power restricted devices such as cellular phones and digital cameras. Therefore, it is challenging to develop hardware algorithms that find connected components using the FPGA with a small internal storage.

Our hardware component labeling algorithm implemented in the FPGA works as follows: The binary image of $m \times n$ (m rows and n columns) is provided to the FPGA such that a pixel is given to the FPGA in every clock cycle in raster order. Hence, $m \times n$ pixels of the binary image are provided to the FPGA in mn clock cycles. In the same time, the FPGA start outputting the resulting label of each pixel in every clock cycle in raster order. Let L denote the latency of our component labeling algorithm, that is, the FPGA outputs the label of the first pixel in L clock cycles after the first binary pixel is provided. Clearly, the total computing time is $mn + L$. Figure 2 illustrates a process of the above execution for an input image of $m \times n$.

The main advantage of our hardware component labeling algorithm is low latency L, that is, L is much smaller than mn. In other words, the label of the first pixel is computed before the last of the input binary image is given to the FPGA as illustrated in Figure 2. Note that, if this is the case, the component labeling for arbitrary binary image is not possible. For example, suppose that we need to find the connected component of the binary image illustrated in Figure 3. Clearly, pixels A and B are in the same connected component, they must be assigned the same label. However, it is impossible to determine if they are in the same connected component before reading the *critical row*, which is the lowest row containing the shortest path between A and B. Since three rows must be read before the label of B is output, the latency must be at least $3n$ to assign the same label to A and B.

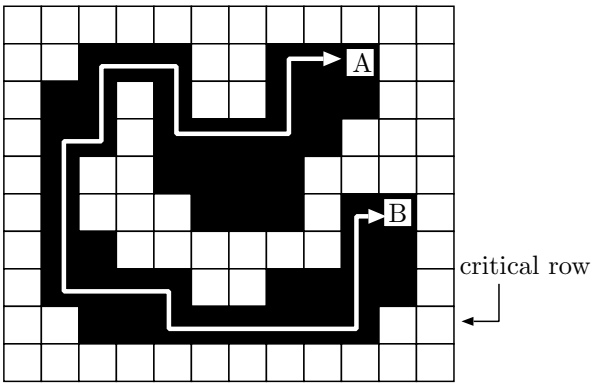


Figure 3: An example of 3-concave binary image.

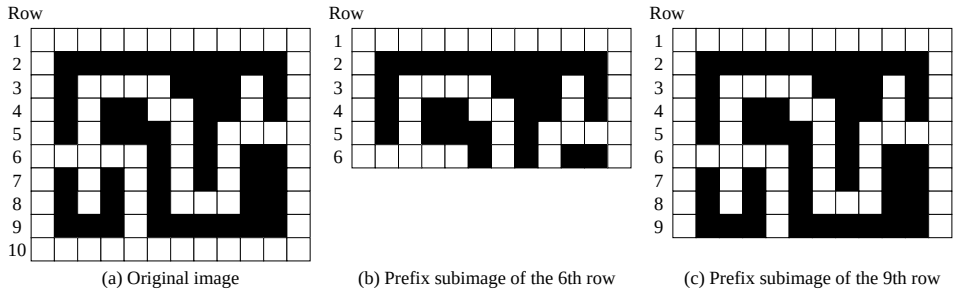
So we need to restrict the binary image to complete correct component labeling in low latency to obtain the correct labeling.

The second contribution of our paper is to introduce a class of binary images called *k-concave* in which the connected component can be done in low latency. A binary image is k-concave if the label of each pixel can be determined using all the pixels on or above the pixel and k rows below it. More specifically, if a binary image is k-concave, then for any two pixels p and q in the same connected component, there exists a path going through the component that are on or above these pixels and below k rows of them. We call such path a *k-concave path.* Figure 3 illustrates an example of 3-concave binary image. As illustrated in the figure, there exists a 3-concave path between pixels A and B. The reader should have no difficulty to confirm that there exists a 3-concave path for any pair of two pixels in the connected component. Thus, the binary image in Figure 3 is 3-concave. However, this binary image is not 2-concave, because there exists no 2-concave path between A and B.

From the theoretical point of view, the latency of our hardware connected component algorithm is optimal. It should be clear that, from the definition of k-concave binary images, the latency cannot be smaller than kn to complete the component labeling for them. Our component labeling hardware algorithm is designed to complete the labeling for k-concave binary image with latency approximately $(2k+2)n$. Therefore our hardware algorithm is *latency optimal* in the sense that the latency is within a constant factor of its lower bound.

Another important merit of our hardware algorithm is to use few internal storage. Our hardware algorithm for k-concave binary image uses the internal storage of size approximately $O(kn)$ words. More specifically, the internal storage are used as a frame buffer that stores the latest $2k + 2$ rows of the binary image and some $O(1)$ words work data of pixels in these $2k + 2$ rows.

We have also implemented our hardware component algorithm in the PCI board with an Altera Stratix FPGA. We describe our hardware algorithm using Verilog HDL and the logic is synthesized using Altera Quartus II design tool. We



(a) Original image (b) Prefix subimage of the 6th row (c) Prefix subimage of the 9th row

Figure 4: Prefix subimage.

confirmed that our hardware component labeling algorithm works correctly in the board, and evaluated the performance. The implementation result shows that for a 10-concave binary image of 2048×2048, our connected component labeling algorithm runs in approximately 70ms and its latency is approximately 750μs.

This paper is organized as follows: Section 2 presents an overview of our connected component labeling algorithm. Section 3 presents the details of it. In Section 4, we show a hardware algorithm of connected component labeling for our connected component labeling. In Section 5, we evaluate the performance of our labeling hardware implementations. Finally, Section 6 is a brief conclusion.

2. Overview of our connected component labeling algorithm

In this section, we present an overview of our connected component labeling algorithm. Before the overview, we introduce some definitions about a subimage of an input binary image and connected components in the subimage. Let *a prefix subimage* of a particular row denote a subimage which consists of the row and the above. For example, Figure 4(b) shows the prefix subimage of the 6th row of Figure 4(a). Clearly, the prefix subimage of the m-th row (i.e. the bottom row) is the whole image. We call connected components in the prefix subimage *prefix connected components*. In addition, let *a k-prefix subimage* of a particular row denote a subimage which consists of rows on or above it and k rows below it. From the definition, generally, it is clear that a k-prefix subimage of the p-th row and a prefix subimage of the $(p+k)$-th row are equivalent. For example, Figure 4(c) shows the prefix subimage of the 9th row of Figure 4(a) and also represents the 3-prefix subimage of the 6th row. Also, connected components in the k-prefix subimage are called *k-prefix connected components*. The readers should have no difficulty to confirm that, if a binary image is k-concave, two pixels are connected if and only if they are connected in the k-prefix subimage. Thus, we have,

Theorem 1 *The k-prefix connected components of a k-concave image are equivalent to the connected components of the whole image.*

According to this theorem, we can obtain the connected component labeling for the k-concave image by performing our connected component labeling method for its k-prefix subimage.

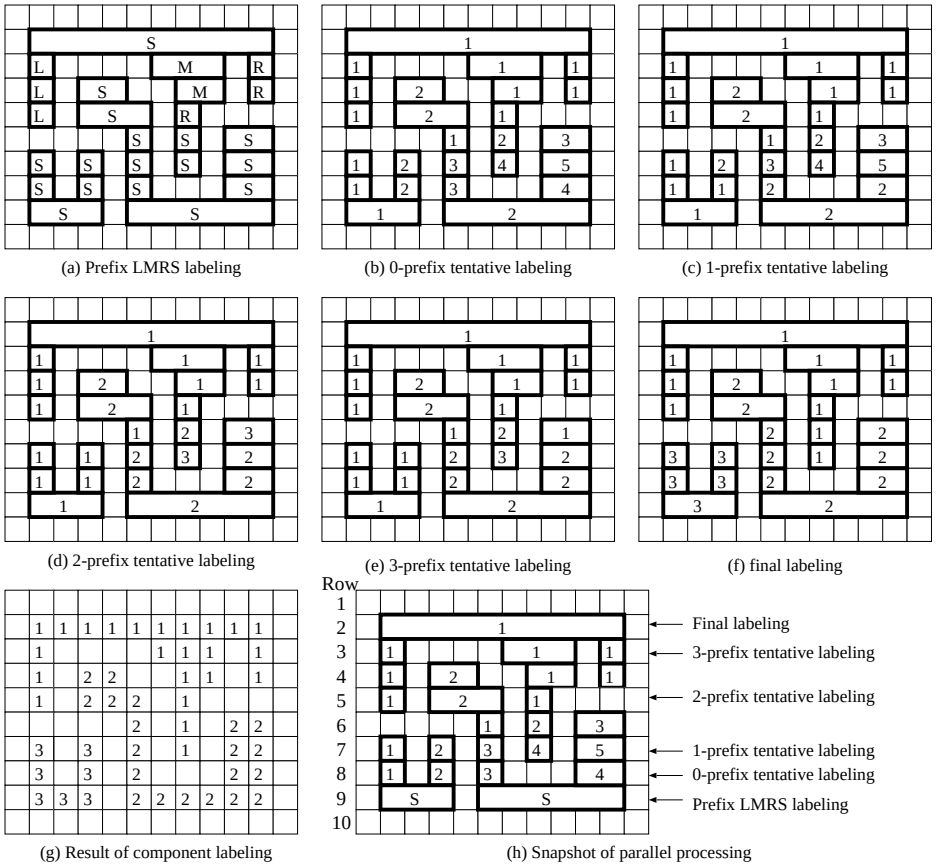


Figure 5: An example of our connected component labeling for a 3-concave binary image of Figure 4(a).

To obtain connected component labels for k-concave images, we assign intermediate labels; *prefix LMRS labeling*, *the i-prefix tentative labeling* for $i = 0, 1, 2, \ldots, k$ and *final labeling*. Figure 5 shows an example of our connected component labeling for a 3-concave binary image. We briefly show the idea to perform our labeling as follows.

1. The prefix LMRS labeling: In the prefix LMRS labeling, each run, which is a maximal sequence of consecutive black pixels, is assigned one of the four symbols L(left), M(middle), R(right), and S(single) (Figure 5(a)). Prefix LMRS labels in each row are assigned for the prefix subimage as follows; if a prefix connected component has a single run in the list, the run is labeled by S. If a prefix connected component has more than two runs in the list, the leftmost and the rightmost runs are labeled by L and R, respectively, and the other runs between them are labeled by M. The prefix LMRS labeling is performed from right to left with a stack in raster scan order by referring the assigned

labels to the upper row.

2. The i-prefix tentative labeling for $i = 0, 1, 2, \ldots, k$: First, each prefix connected component is assigned an integer as *a 0-prefix tentative label* (Figure 5(b)). The 0-prefix tentative labels are assigned from left to right in each row such that if two runs that are in a particular row belong to the same prefix connected component, they are assigned by a same 0-prefix tentative label. If not, they are assigned by distinct labels (Figure 6). Therefore, the

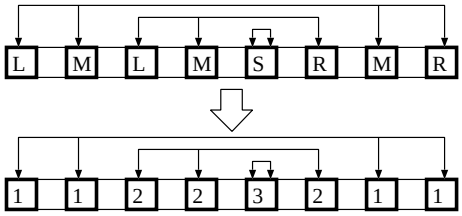


Figure 6: An example of 0-prefix tentative labeling

0-prefix tentative labeling can be done simply by scanning the assigned prefix LMRS labels using a stack. Let us consider the 0-prefix tentative algorithm in a particular row. The algorithm scans the row from left to right and assigns the 0-prefix tentative labels to runs, as follows.

- If a run assigned S is found, the run is assigned a new label.
- If a run assigned L is found, the run is assigned a new label and its label is pushed to the stack.
- If a run assigned M is found, the run is assigned a label in the stack top.
- If a run assigned R is found, the run is assigned a label in the stack top and the stack top is popped from the stack.

The above operation for each row is performed from the top row to the bottom row, and then the algorithm completes the 0-prefix tentative labeling. Note that the 0-prefix tentative labels are assigned labels from 1 in each row, that is, the leftmost run that is assigned S or L is assigned label 1.

Next, the 1-prefix tentative labels are assigned by updating the 0-prefix tentative labels from the 0-prefix tentative labels in the lower row such that if two or more runs, which are in each row, are adjacent to runs with a label in the lower row, a same label is assigned to those. For example, in Figure 7, the 6th and 7th runs from left in the p-th row are adjacent to a run in the lower row, and then the runs which are assigned the same labels as the 6th and 7th runs in the p-th row are assigned a same run. Then, we obtain the 1-prefix tentative labels in each row that are assigned for the 1-prefix subimage.

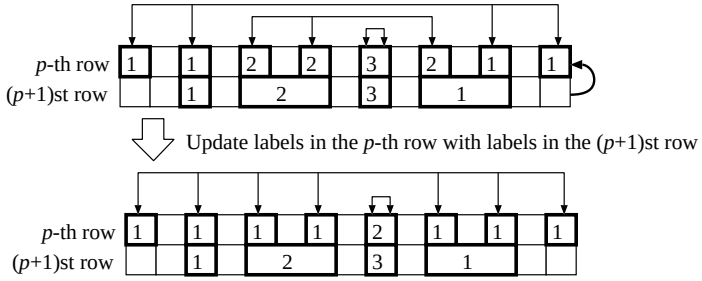


Figure 7: An example of 1-prefix tentative labeling

Similarly, executing the above update 2 times from two rows below it, we obtain 2-prefix tentative labels in each row that are assigned for the 2-prefix subimage (Figure 8).

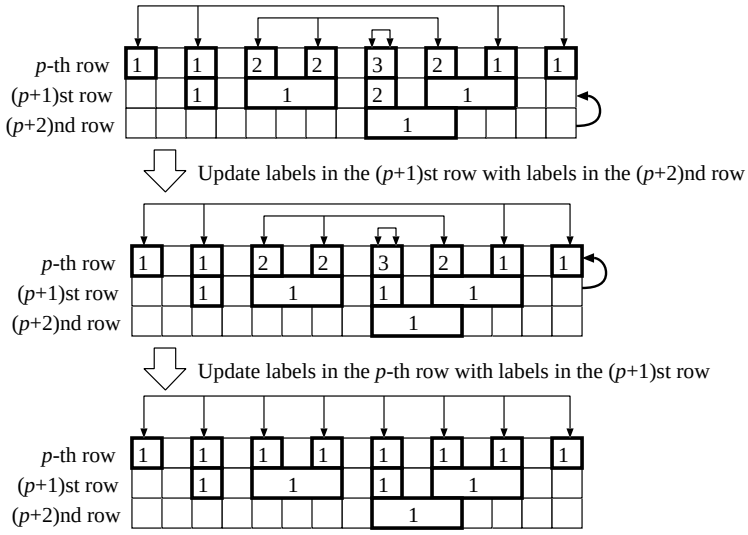


Figure 8: An example of 2-prefix tentative labeling

To obtain k-prefix tentative labels, the above update is performed k times from k rows below it in each row. From Theorem 1, after k-prefix tentative labeling for k-concave image, if there are two or more runs that are assigned the same label in a particular row, they belong to the same connected components. If not, they belong to distinct connected components. For example, to perform 3-prefix tentative labeling, 0-prefix tentative labeling, 1-prefix tentative labeling, 2-prefix tentative labeling and 3-prefix tentative labeling are executed (Figure 5(b)-(e)).

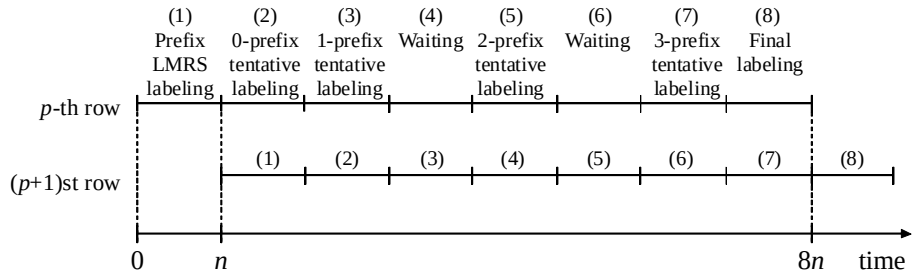


Figure 9: Pipeline processing row by row for a 3-concave binary image.

3. The final labeling: In the final labeling, each run is assigned connected component label using the k-prefix tentative labels. By scanning the k-prefix tentative labels, the final labels are assigned in raster scans order such that if a run that is adjacent to the upper row, the same label is assigned to the run and if not, a new label is assigned. Then, we obtain the connected component labels by outputting the labels pixel by pixel (Figure 5(g)).

Hereafter, let the connected component labeling for a k-concave binary image *k-concave labeling*.

In our hardware algorithm for our connected component labeling, the above methods are performed by pipeline processing row by row in parallel from the top row to the bottom row. Figure 9 shows the pipeline processing for the p-th row and the $(p+1)$-th row in a 3-concave binary image. For a 3-concave image, the pipeline processing consists of 8 stages row by row as shown in the figure. Note that in the 4th and 6th stage, the process wait for the results for the $(p+1)$-th row. Figure 5(h) shows an assignment of the parallel processing when the final labeling for the 4th row has been completed. The hardware can execute the above operation for each pixel in 1 clock cycle. Then, the computing time for each row is approximately $8n$ clock cycles for 3-concave image. In addition, the latency, which is time from the first pixel is given to the first result of the final labeling, is also approximately $8n$ clock cycles, and the computing time for the whole image is approximately $(m+8)n$ clock cycles. More generally, for k-concave image, the latency is approximately $(2k+2)$ and the computing time for the whole image is approximately $(m+2k+2)n$ clock cycles.

3. The details of our connected component labeling algorithm

As shown in Section 2, to obtain connected component labels for k-concave images, we assign intermediate labels; *prefix LMRS labeling*, *i-prefix tentative labeling* for $i = 0, 1, 2, \ldots, k$ and *final labeling*. This section shows the details of these methods.

3.1. The prefix LMRS labeling

Recall that in the prefix LMRS labeling, each run is assigned one of the four symbols L, M, R and S for the prefix subimage in each row.

We present the detail of the algorithm of the prefix LMRS labeling. The prefix LMRS labels are assigned in raster scan order. Let us consider an assignment of the prefix LMRS labels in the p-th row. We assume that it completes the assignment of the prefix LMRS labels for the $(p-1)$-th row. To assign the labels, the followings are done, as follows.

Algorithm 1 (Prefix LMRS labeling):

Step 1: For each row p, compute the followings by scanning the p-th row and the $(p-1)$-th row from left to right.

(1-1) Use a stack to store the information of each prefix connected component in the two rows. Each element of the stack consists of three values, *left*, *right* and *nest*, as follows:

- *left*: a sequential ID of the leftmost run that belongs to a prefix connected component in the p-th row.
- *right*: a sequential ID of the rightmost run that belongs to a connected component in the p-th row.
- *nest*: the number of nested connections of the left side in the p-th row.

The sequential ID shows the number of runs from left in a particular row.

For example, the sequential ID of the leftmost run is 1 and the sequential ID of the second run from left is 2. In addition, if there are no runs that belongs to a prefix connected component in the p-th row, let *left* and *right* be ϕ which represents a special symbol. If there is one run that belongs to a prefix connected component in the p-th row, let *left* and *right* be its sequential ID. When a run which belongs to a prefix connected component in the p-th row is found, each element of the stack top is updated like the above.

According to this operation, the value of *left* and *right* shows the number of runs that belong to the prefix connected component in the p-th row as follows; if $left = \phi$, there exist no runs in the prefix connected component, if $left \neq \phi$ and $left = right$, there exists one run in the prefix connected component, and if $left \neq right$, there exist two or more runs in the prefix connected component.

Then, a new element is pushed to the stack when the leftmost pixel of the prefix connected component in the $(p-1)$-th and p-th row is found, and the top entry of the stack is popped when the rightmost pixel of connected component in the $(p-1)$-th and p-th row is found so that the top entry of the stack is always an element that is concerned with the scanned prefix connected component at present.

(1-2) Assign a prefix LMRS label to the run when a run in the p-th row which belongs to the scanned prefix connected component is found, as follows:

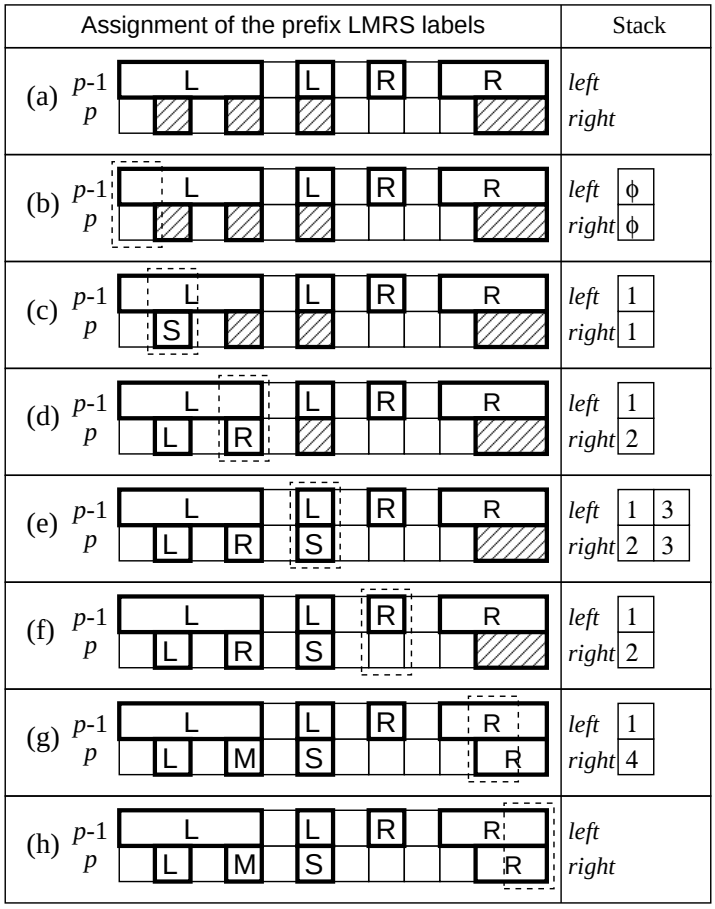


Figure 10: An example of a process of an assignment prefix LMRS labels

- If $left = \phi$, the run is assigned S.
- If $left \neq \phi$ and $left = right$, the run is assigned R and the run at *left* is assigned L.
- If $left \neq right$, the run is assigned R and the run at *right* is assigned M.

After this assignment, each element of the stack top is updated. Note that every run in the top row is assigned S because every prefix connected component in the top row consists of a single row.

Figure 10 shows an example of a process of an assignment prefix LMRS labels in the p-th row. We assume that the assignment of the prefix LMRS labels to the $(p-1)$-th row is completed (Figure 10(a)). To assign prefix LMRS labels, the p-th row and $(p-1)$-th row are scanned from left to right (Figure 10(b)-(h)). First, a run which is assigned L in the $(p-1)$-th row is found, then the element ($left = \phi$ and $right = \phi$) is pushed to the stack (Figure 10(b)). Next, a run in the p-th row is

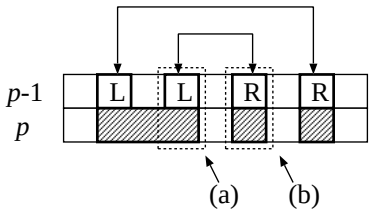


Figure 11: An example of the nested structure that connects at the left side

found and it is adjacent to the run in $(p-1)$-th row. Both elements of the stack top are ϕ, then it shows that the run is a first run in the prefix connected component. Therefore, the element of the stack top is updated to $left = 1$ and $right = 1$ and the run is assigned S (Figure 10(c)). After that, a run in the p-th row is found and it is adjacent to the run in $(p-1)$-th row. The elements of the stack top are the same, then the run is a second run in the prefix connected component. Therefore, the element of the stack top is updated to $right = 2$ and the run which is shown by *left* is assigned L and the scanned run is assigned R (Figure 10(d)). Next, a run which is assigned L in the $(p-1)$-th row and a run which is adjacent to it are found. These runs belong to a new prefix connected component, then the element ($left = 3$ and $right = 3$) is pushed to the stack and the run in the p-th row is assigned S (Figure 10(e)). After that, a run which is assigned R in the $(p-1)$-th row is found, it is the leftmost run of the prefix connected component then the element of the stack top is popped (Figure 10(f)). Next, a run in the p-th row is found and it is adjacent to the run in $(p-1)$-th row. The elements of the stack top are different, then the run which is shown by *right* is assigned M and the scanned run is assigned R and the element of the stack top is updated to $right = 4$ (Figure 10(g)). Finally, the rightmost run of the prefix connected component is found, then the stack top is popped (Figure 10(h)).

During scan, two or more prefix connected components in the upper row that consists of a nested structure combined as the inside one connects to the left of outside one in the p-th row (Figure 11). In this case, the stack is not pushed when the inside prefix connected component is found. Then, when the rightmost run of the inside prefix connected component is found, the stack must not be popped. Therefore, we use *nest* that is an element of the stack entry. When the above case is found, *nest* of the stack top is incremented and is decremented instead of popping. For example, when a nest that connects at the left side in the p-th row is found, *nest* of the stack top increments (Figure 11(a)). When the rightmost run of the nest is found, *nest* of the stack top is decremented (Figure 11(b)).

3.2. The i-prefix tentative labeling for $i = 0, 1, 2, \ldots, k$

Recall that the i-prefix tentative labeling assigns the i-prefix tentative labels such that if two runs in a particular row belong to the same connected component in the i-prefix subimage, they are assigned by a same i-prefix tentative label. If not,

they are assigned by distinct labels. In this section, we show the algorithm for the i-prefix tentative labeling for the case $i = 0$ and $i > 0$ separately, as follows.

Algorithm 2: (The k-prefix tentative labeling)

Step 1: (The 0-prefix tentative labeling)
Assign a same 0-prefix tentative label to two or more runs that are in a particular row if they belong to the same connected component in the prefix subimage, If not, assign distinct labels to them. This means that in the 0-prefix tentative labeling, each prefix connected component in each row is assigned a unique label. Therefore, the 0-prefix tentative labeling can be done simply by scanning the assigned prefix LMRS labels using a stack. For each row p, assign the 0-prefix tentative labels by scanning from left to right in the followings.

case 1: (a run assigned S is found)
Assign a new label to the run.

case 2: (a run assigned L is found)
Assign a new label to the run and push its label to the stack.

case 3: (a run assigned M is found)
Assign a label in the stack top.

case 4: (a run assigned R is found)
Assign a label to the run and pop the stack top from the stack.

Note that the 0-prefix tentative labels are assigned labels from 1 in each row, that is, the leftmost run that is assigned S or L is assigned label 1. In the 0-prefix tentative labeling, the computing time for each pixel is $O(1)$, and the computing time for whole image that can be done for an input image of $m \times n$ is mn because the 0-prefix tentative labels can be determined as soon as each run and its prefix LMRS labels are given.

Step 2: (The i-prefix tentative labeling ($i > 0$))
Assign i-prefix tentative labels using the $(i-1)$-prefix tentative labels using a table such that when a $(i-1)$-prefix tentative label in the $(p+1)$-th row is given as the table index, the entry of the table is the i-prefix tentative label that is assigned to the adjacent one or more runs in the p-th row. For each row $p(> 1)$, assign the i-prefix tentative labels from the bottom row to the top row as follows.

(1) Initialize every table entry to 0.

(2) Assign labels using the table as follows.

case 1: (a run is assigned by S or L is found)
Assign a new label which starts from 1 to its prefix connected component since the run is the leftmost of the prefix connected component.

case 2: (a vertically adjacent run in the $(p+1)$-th row is found)
If the entry of the table where the index is the label of adjacent run is 0, that is, it is the first time to meet an adjacent run in the $(p+1)$-th row, store the label to the entry. If not, that is the entry of the table where the index is the label of adjacent run is not 0, assign the label in the entry to the prefix connected component in the p-th row.

To obtain the assignment of the k-prefix tentative labels from 0-prefix labels, we perform 1-prefix tentative labeling, 2-prefix tentative labeling, ..., k-prefix tentative labeling, that is, the above process is executed k times. The computing time of once update for an input image of $m \times n$ is mn such as 0-prefix tentative labeling. Therefore, it takes kmn time to perform the k-prefix tentative labeling for the image that is already assigned by 0-prefix tentative labels.

3.3. The final labeling

We introduce the assignment of the connected component labeling using the tentative labels. In this algorithm, the labels are assigned to each pixel in raster scan order as follows.

Algorithm 3: (The final labeling)
For each row p, compute the followings from the top row to the bottom row.

Step 1: Make a table to convert the tentative labels to the connected component labels. This can be done as follows.

(1) Trace the prefix connected components using the assigned prefix LMRS labels.

(2) Store assigned connected component labels for each component with a stack.

(3) Check whether the connected components is adjacent to any pixels that are in the upper row. If a connected component is not adjacent to any runs, update the table so as to convert the tentative label of the prefix connected component to a new label.

Step 2: Convert the k-prefix tentative labels to the connected component labels by referring the table and assign the connected component label to the scanned pixel.

4. Hardware Algorithm for our connected component labeling

In this section, we show a hardware algorithm for our connected component labeling (i.e. k-concave labeling).

Recall that to perform our connected component labeling, the following processes are performed for an input image in this order; the prefix LMRS labeling, the 0-prefix tentative labeling, the 1-prefix tentative labeling, ..., the k-prefix tentative labeling and the final labeling. Each labeling method can be done in raster scan order. The prefix LMRS labeling, the 1-prefix tentative labeling, ..., the k-prefix tentative labeling and the final labeling refer their prior process's results in only the upper or lower row. Therefore, each labeling method does not need to wait its prior process's results for the whole image. According to the above reason, the hardware algorithm performs pipeline processing in parallel row by row from top to bottom in the above order of labeling methods. For each row, namely, the prefix LMRS labeling, the 0-prefix tentative labeling, the 1-prefix tentative labeling, ..., the k-prefix tentative labeling and the final labeling are performed and each labeling method executes concurrently.

Note that in the i-prefix tentative labeling ($i > 0$), the results for the individual row can be also output until the process for the end of the row is finished, however it refer the $(i-1)$-prefix labels in the lower row. Because the parallel processing is performed from the top row to the bottom row, then in our connected component labeling, to assign labels to a particular row, the process must wait the assignment of $(i-1)$-prefix tentative labels to the lower row. On the other hand, in the prefix LMRS labeling and the final labeling, the results for the individual row can be output until the process for the end of the row is finished. Namely, a result for a particular row is used in the process for the next row. Also, in the 0-prefix tentative labeling the results for each input pixel can be output as soon as its own process is finished, and the process for the next row can be done without its result.

Figure 12 shows our hardware implementation for the k-concave labeling that performs the above processes. It consists of the circuits for the prefix LMRS labeling, the i-prefix tentative labeling for $i = 0, 1, 2, \ldots, k$ and the final labeling that are arranged with cascade connection and perform pipeline processing, and two RAMs that supply the subimage and its prefix LMRS labels to these circuits. We explain each circuit, as follows:

- The circuit for the prefix LMRS labeling : The circuit for the prefix LMRS labeling assigns the prefix LMRS labeling to each run. It has an input for pixel data from the input image and outputs its prefix LMRS labels. It uses a $\lceil n/2 \rceil$ words storage to refer the assigned prefix connected components to the upper row and a $\lceil n/4 \rceil$ words stack to store the information of the adjacent prefix connected components. It takes 1 clock cycle for each input, however the labels for each row cannot be determined until the operation for the rightmost pixel is finished. Accordingly, the latency is n clocks cycles and the computing time that the prefix LMRS labeling can be done for an input image mn clock cycles.

- The circuit for the i-prefix tentative labeling for $i = 0, 1, 2, \ldots, k$: The circuit for the i-prefix tentative labeling for $i = 0, 1, 2, \ldots, k$ assigns the 0-prefix tentative labels, 1-prefix tentative labels, 2-prefix tentative labeling, ..., k-

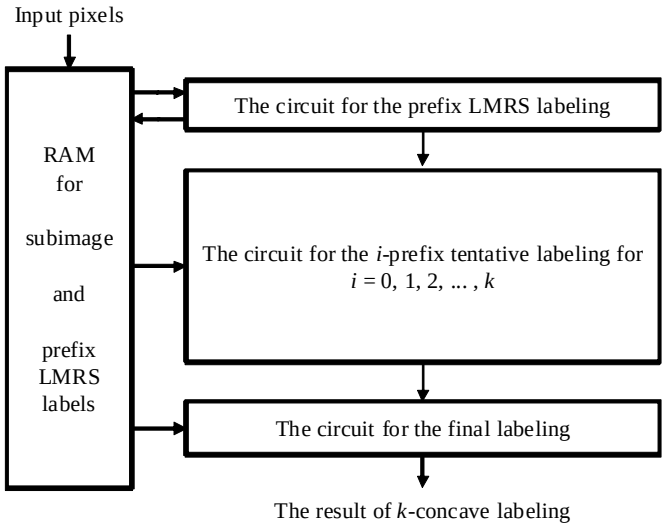


Figure 12: Our hardware implementation for k-concave labeling.

prefix tentative labels to each run. It has an input for pixel data, the prefix LMRS labels and it outputs its k-prefix tentative labels. In addition, as illustrated in Figure 13, it consists of the circuit for 0-prefix tentative labeling and k circuits for the label rewriting.

The circuit for the 0-prefix tentative labeling assigns the 0-prefix tentative labels to each prefix connected component. It has inputs for pixel data and the prefix LMRS labels from the circuit for the prefix LMRS labeling and outputs its 0-prefix tentative labels. Because it takes 1 clock cycle to assign labels for each input and the labels can be determined as soon as the input is given, then the latency is 1 clock cycle and the computing time that the prefix LMRS labeling can be done for an input image mn clock cycles.

The circuit for the label rewriting rewrites labels by referring the labels in the lower row. It uses a $\lceil n/2 \rceil$ words storage to refer the assigned prefix connected components and a $\lceil n/4 \rceil$ words stack to store the information of the prefix connected components. It takes 1 clock cycle for each input pixel, however the labels for each row cannot be determined until the operation for the rightmost pixel is finished. Then, the latency is n clock cycles and the computing time that the circuit for a row can be done for an input image mn clock cycles.

As a result, the circuit for the k-prefix tentative labeling that consists of the circuit for 0-prefix tentative labeling k circuits for the label rewriting uses a $k\lceil n/2 \rceil$ words storage to refer the assigned prefix connected components to the upper row and a $k\lceil n/4 \rceil$ words stack to store the information of the adjacent prefix connected components. The latency is $2kn$ clock cycles and the computing time that the prefix LMRS labeling can be done for an input

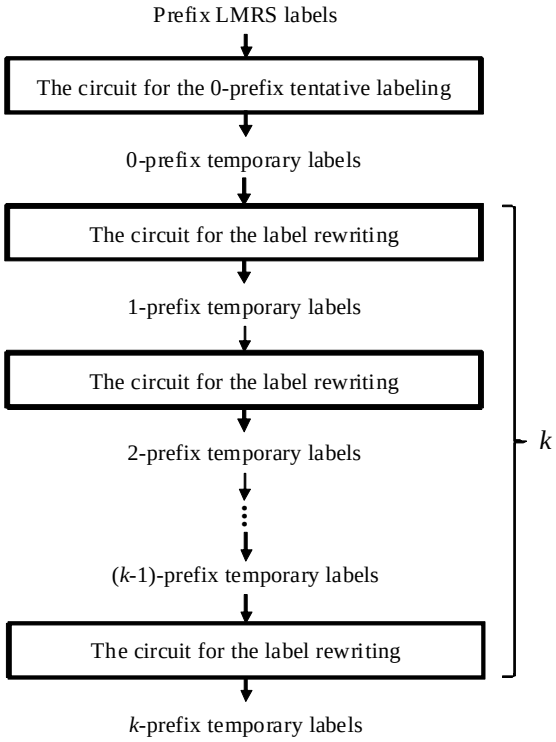


Figure 13: Hardware implementation for i-prefix tentative labeling for $i = 0, 1, 2, \ldots, k$.

image mn clock cycles.

- The circuit for the final labeling: The circuit for the final labeling assigns the component labels to each pixel. It has an input for pixel data, the prefix LMRS labels and the k-prefix tentative labels and it outputs its k-prefix tentative labels. It uses a $\lceil n/2 \rceil$ words storage to refer the assigned prefix connected components to the upper row and a $\lceil n/4 \rceil$ words stack to store the information of the adjacent prefix connected components. It takes 1 clock cycle for each input and the labels for each row cannot be determined until the operation for the rightmost pixel is finished. Accordingly, the latency is n clock cycles and the computing time that the prefix LMRS labeling can be done for an input image mn clock cycles.

- RAM: The two buffers stores the subimage data and its prefix LMRS labels and supplies to these circuits. Each buffer consists of an array of RAMs. Each buffer has $2k+2$ banks to store and supply data row by row. More specifically, the RAMs are used to store the latest $2k + 2$ rows of the input binary image

Table 1: Latency of each circuit for k-concave labeling.(n:width of the input image)

Circuit	Latency[clock cycles]
The prefix LMRS labeling	n
The i-prefix tentative labeling for $i = 0, 1, 2, \ldots, k$	$2kn$
The final labeling	n
Total	$(2k + 2)n$

and their prefix LMRS labels.

According to using the above circuits, out hardware works as follows: The binary image of $m \times n$ is provided to the hardware such that this input is given to the hardware in every clock cycle in raster order. Hence, this input is sent to the hardware in mn clock cycles. On the other hand, the latency is $(2k + 2)n$ clock cycles as shown in Table 1 because each circuit cannot start the process until each prior process starts to output the results. Therefore, the total computing time is $(m + 2k + 2)n$ clock cycles.

This hardware algorithm for k-prefix tentative labeling uses the internal storage of size approximately $O(kn)$ words. More specifically, the internal storage are used as a frame buffer that stores the latest $2k + 2$ rows of the binary image and $O(n)$ words tentative data of pixels in these $2k + 2$ rows.

5. Performance Evaluations

We have implemented our hardware algorithm for k-concave labeling in the PCI board with an Altera Stratix family FPGA (EP1S25, speed grade 7, typical 2.5 million gates with 2 Mbits embedded memory), and have evaluated the performance. The logic is synthesized using Altera Quartus II design tool, and we have synthesized the implementations of 0-concave labeling, 1-concave labeling,. . . , 10-concave labeling, and executed them on the FPGA. These circuits can perform for the image of the size 2048×2048.

Table 2 shows the result of the number of used logic elements, the size of used internal RAM and the maximum frequency of the synthesis of the k-concave labeling circuits for various k's. According to the table, the hardware implementation for 10-concave labeling runs in 60MHz. Therefore, for a 10-concave image of 2048×2048, our 10-concave labeling circuit runs in approximately 70ms and its latency is approximately 750μs. The result confirms that this hardware implementation has enough performance as a part of the image recognition system. Moreover, as k gets larger, the number of used logic components and the size of used internal RAMs are increasing in a linear fashion without excessive decrease of frequency. Consequently, it is possible to use the intended hardware implementation by selecting the appropriate value of k for used device or target application.

6. Concluding Remarks

Table 2: Result of synthesis of the our hardware for k-concave labeling

k	Logic Elements	Inside RAM [bit]	Maximum Frequency [MHz]
0	1596	53760	72.66
1	2528	89344	63.52
2	3479	124928	61.32
3	4420	160512	64.07
4	5358	196096	60.00
5	6284	231680	60.87
6	7218	267264	61.47
7	8154	302848	61.47
8	9092	338432	62.02
9	10028	374016	60.71
10	10964	409600	61.61

We have presented a hardware connected component labeling algorithm for k-concave binary images designed and implemented in FPGA. The hardware is performed by row-based pipeline processing in parallel. Pixels of a binary image are given to the FPGA in raster order, and the resulting labels are also output in the same order. The advantage of our labeling algorithm is small latency and to use a small internal storage of the FPGA. We have implemented our hardware labeling algorithm in an Altera Stratix Family FPGA, and evaluated the performance. The implementation result shows that for a 10-concave binary image of 2048×2048, our connected component labeling algorithm runs in approximately 70ms and its latency is approximately 750μs. The result confirms that this hardware implementation has enough performance as a part of the image recognition system.

References

1. Azriel Rosenfeld, "Connectivity in digital pictures," *Journal of the ACM*, vol. 17, no. 1, pp. 146–160, 1970.
2. Azriel Rosenfeld and Avinash C. Kak, *Digital Picture Processing*, Academic Press, Inc., 2nd edition, 1982.
3. Xue-Dong Tian, Hai-Yan Li, Xin-Fu Li, and Li-Ping Zhang, "Research on symbol recognition for mathematical expressions," in *Proceedings of the First International Conference on Innovative Computing, Information and Control*, 2006, vol. 3, pp. 357–360.
4. Azriel Rosenfeld and John L. Pfaltz, "Sequential operations in digital picture processing," *Journal of the ACM*, vol. 13, no. 4, pp. 471–494, 1966.
5. Tetsuo Hattori, "A high-speed pipeline processor for regional labelling based on a new algorithm," in *International Conference on Pattern Recognition*, Jun 1990, vol. 2, pp. II: 494–496.
6. Ramana V. Rachakonda, Peter M. Athanas, and A. Lynn Abbott, "High-speed region detection and labeling using an fpga-based custom computing platform," *Proc. Field Programmable Logic and Applications (FPL) (Lecture Notes in Computer Sci-*

ence), vol. 975, pp. 86–93, 1995.

7. Chris John Nicol, "A systolic approach for real time connected component labeling," *Computer Vision and Image Understanding*, vol. 61, no. 1, pp. 17–31, 1995.
8. Luigi di Stefano and Andrea Bulgarelli, "A simple and efficient connected components labeling algorithm," in *Image Analysis and Processing, 1999. Proceedings. International Conference on*. 1999, pp. 322–327, IEEE Computer Society.
9. Frederic Planque, Ivan C. Kraljic, and Yvon Savaria, "Mapping irregular algorithms in a custom computing image processing framework," in *2000 MAPLD International Conference*, 2000.
10. Jung-Me Park, Carl G. Looney, and Hui-Chuan Chen, "Fast connected component labeling algorithm using a divide and conquer technique," *University of Alabama, CS Dept., University Nevada, CS Dept. http://cs.ua.edu/TechnicalReports/TR-2000-04.pdf*, 2000.
11. Vikrant Khanna, Phalguni Gupta, and C. J. Hwang, "Finding connected components in digital images," in *Proceedings of the International Conference on Information Technology: Coding and Computing*, 2001, pp. 652–656.
12. Christian Schmidt and Andreas Koch, "Fast region labeling on the reconfigurable platform ace-v," in *Proceedings of International Conference on Field-Programmable Logic, Lisbon (PT), September 2003*. 2003, Springer.
13. Khaled Benkrid, S. Sukhsawas, Danny Crookes, and Abdsamad Benkrid, "An FPGA-based image connected component labeller," in *Field-Programmable Logic and Applications*, Peter Y. K. Cheung, George A. Constantinides, and José T. de Sousa, Eds. 2003, vol. 2778 of *Lecture Notes in Computer Science*, pp. 1012–1015, Springer.
14. Miroslaw Jablonski and Marek Gorgon, "Handel-C implementation of classical component labelling algorithm," *Digital System Design, 2004. Euromicro Symposium on*, pp. 387–393, 2004.
15. Kesheng Wu, Ekow Otoo, and Arie Shoshani, "Optimizing connected component labeling algorithms," in *Medical Imaging 2005: Image Processing. Proceedings of the SPIE, Volume 5747*, Jan. 16 2005, vol. 5747, pp. 1965–1976.
16. Kenji Suzuki, Isao Horiba, and Noboru Sugie, "Linear-time connected-component labeling based on sequential local operations," *Computer Vision and Image Understanding*, vol. 89, no. 1, pp. 1–23, 2003.
17. PE Danielsson, "An improved segmentation and coding algorithm for binary and nonbinary images," *IBM Journal of Research and Development*, vol. 26, no. 6, pp. 698–707, 1982.
18. Fu Chang, Chen, Chun-Jen Chen, and Chi-Jen Lu, "A linear-time component-labeling algorithm using contour tracing technique," *Computer Vision and Image Understanding*, vol. 93, no. 2, pp. 206–220, 2004.
19. Robert Cypher, Jorge L C Sanz, and Lawrence Snyder, "Algorithms for image component labeling on simd mesh-connected computers.," *IEEE Trans. Computers*, vol. 39, no. 2, pp. 276–281, 1990.
20. Hussein M. Alnuweiri and Viktor K. Prasanna, "Parallel architectures and algorithms for image component labeling," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 10, pp. 1014–1034, 1992.
21. Poras T. Balsara and Mary Jane Irwin, "Intermediate-level vision tasks on a memory array architecture," *Machine Vision and Applications*, vol. 6, no. 1, pp. 50–65, 1993.

22. John Greiner, "A comparison of parallel algorithms for connected components," in *Proceedings of the 6th Annual ACM Symposium on Parallel Algorithms and Architectures*, Cape May, New Jersey, June 27–29, 1994, SIGACT and SIGARCH, pp. 16–25.
23. Prabir Bhattacharya, "Connected component labeling for binary images on a reconfigurable mesh architecture," *Journal of Systems Architecture*, vol. 42, no. 4, pp. 309–313, 1996.
24. David A. Bader and Joseph JáJá, "Parallel algorithms for image histogramming and connected components with an experimental study," *Journal of Parallel and Distributed Computing*, vol. 35, no. 2, pp. 173–190, 15 June 1996.
25. Alina N. Moga and Moncef Gabbouj, "Parallel image component labeling with watershed transformation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 5, pp. 441–450, May 1997.
26. Massimo Maresca, Hungwen Li, and M. Lavin, "Connected component labeling on polymorphic torus architecture," in *Computer Vision and Pattern Recognition, 1988. Proceedings CVPR'88., Computer Society Conference on*, 1988, pp. 951–956.
27. Vipin Chaudhary and Jake K. Aggarwal, "Parallel image component labeling for target acquisition," *Optical Engineering*, vol. 37, no. 7, pp. 2078–2090, jul 1998.
28. Wim H. Hesselink, Arnold Meijster, and Coenraad Bron, "Concurrent determination of connected components," *Science of Computer Programming*, vol. 41, no. 2, pp. 173–194, oct 2001.
29. Xue Dong Yang, "Design of fast connected components hardware," *Computer Vision and Pattern Recognition. Proceedings CVPR'88., Computer Society Conference on*, pp. 937–944, 1988.