Proposal and Implementation of the Connected-Component Labeling of Binary Images and Filling Holes for GPGPU

Hiroto Kizuna *v2649861@gmail.com*
Graduate School of Software and Information Science
Iwate Prefectual University (Current affiliation: KDDI corporation)
Takizawa,020-0693,Japan

Hiroyuki Sato *sato_h@iwate-pu.ac.jp*
Graduate School of Software and Information Science
Iwate Prefectual University
Takizawa,020-0693,Japan

Abstract

The connected-component labeling (CCL) is a technique for extracting connected pixels having the same value. It is mainly used for abnormality diagnosis of products, and for extracting noise areas of products. In the extraction of product areas in product diagnosis, a hole filling processing (HFP) is used to complement discolored areas. However, the HFP is inefficient, because the CCL needs to be executed twice in the foreground and background, and half of the threads are idle during each process. In this study, we propose a rewriting method for continuous label IDs with pixel-by-pixel parallelism, and a HFP method using simultaneous CCL of foreground and background. We implemented and evaluated these methods on Jetson TX2. The rewriting process to the continuous label ID is 3.7-13.8 times faster than the conventional method of sequential processing on the CPU, and on average 9.2 times faster. For the HFP using simultaneous CCL, we implemented and verified the conventional method that requires twice the CCL and the proposed method that can extract the foreground and background with one CCL. The performance of the proposed method is about 13-27% better than that of the conventional method. In addition, in the lightweight object detection method that is an application using the proposed method, the facial detection time is about 14 ms, which is about 60 times faster than the conventional method. As a result, the facial detection processing with high computational complexity can be operated practically even on an inexpensive and small processor. The CCL process for GPGPU has little room for optimization, and it has been difficult to achieve higher speeds. However, we focused on wasted idols in the HFP, proposed a method to reduce and supplement them, and realized a faster HFP than the conventional method.

Keywords: Connected Component Labeling, Parallel Processing, GPGPU, Image Processing.

1. INTRODUCTION

Connected-Component Labeling (CCL) [1] is one of the most important kernels in image processing. It is a process that extracts a group of pixels connected with the same pixel value from the input image as a connected component. It is mainly used for diagnosis using images such as diagnosis of defects in production lines, medical image diagnosis using CT images, pest diagnosis of crops and estimation of yield. An application of extracting objects or noise area in an image is a process that fills non-object area (perforation) in the object area (hereinafter referred to as hole filling process: HFP) [2][3][4][5]. The HFP can extract a noise-removed object region by connecting a non-object connected component with a small area to an object connected component. Processing methods such as histogram calculation are used to calculate the area of non-object connected components. Therefore, using the label ID added to the extracted connected components, the histogram bin (array) is accessed and the number of pixels is counted. However, the label ID of the final label image is a non-consecutive number, and the

maximum number can be the number of pixels of the input image. This degrades the locality of the access pattern to the histogram bin and causes cache efficiency to decrease. Furthermore, resource efficiency is poor because it is necessary to secure the histogram bins for the number of pixels of the input image in the memory.

CCL processing is computationally expensive compared with image processing kernels such as Gaussian filters and is likely to become a bottleneck in real-time processing. Furthermore, a general CCL algorithm is not suitable for SIMD (Single Instruction Multiple Data stream) architecture because it does not have simple parallelism and requires a large number of complex branch instructions. Thus performance improvement by parallelization cannot be expected. The HFP requires longer processing time because the connected components are extracted for the both foreground and background.

In general, when processing images on a GPU, one thread is assigned to each pixel. However, there is no need to extract the background when foreground components are extracted, and the foreground when background is extracted. As a result, half of the two CCL are in an idle state, and the processing time is long and the execution performance is degraded.

In this study, we propose a parallel algorithm to rewrite the label IDs of CCL label images to consecutive numbers starting from 1 for the above-mentioned issues. In addition, in order to speed up the HFP, we propose a HFP algorithm using connected components that simultaneously extract the foreground and background components. We use the propagation CCL algorithm [6][7] which is for many-core architectures such as GPGPU proposed by Shibata et al. The implementation environment was Tegra X2 (TX2) [8] which is a small and lightweight embedded SoC with a GPU provided by NVIDIA. This is because it is assumed to be used in factory production lines and in-vehicle applications. In this study, we verified the processing speed and execution efficiency by parallelization on TX2.

Section 2 explains the technologies used, related works, and general application techniques. In section 3, we explain two proposed methods: rewrite algorithm for continuous label ID for GPGPU and filling process using connected components extracted simultaneously foreground and background. Section 4 shows the results of verification, and section 5 concludes.

2. RELATED TECHNOLOGIES

2.1. CCL

2.1.1. Algorithms

CCL targets binary images with True (1) and False (0) values, adds one or more unique label IDs to connected pixel groups with the same pixel value, adds label IDs of 0 to the other pixels, and outputs a label image as shown in FIGURE 1 (d). There are two types of CCL processing algorithms: raster type and propagation type. In this research, we will focus on the extraction of 4 connected components in the vertical and horizontal directions. Two methods are described below.

The raster type sequentially scans from the upper left corner to the lower right of the binary image, and if there are foreground pixels in the two neighboring pixels which are up and left of the target pixel, the smallest label ID in them is added to the target pixel. If no foreground pixel exists, a new label ID is added to the target pixel. If the foreground pixel has the pattern shown in FIGURE 2 (b), different label IDs will be added even to the connected pixels, since the raster type scans from the upper left to the lower right. So, in the raster type, the identity of label IDs is stored using Look-Up Table (LUT), and the label is rewritten after labeling to all pixels is completed. Therefore, the amount of calculation is relatively lower than the propagation type described below, since the connected components can be extracted by two scans. However, there is no simple parallelism, since it scans sequentially from the upper left to the lower right.

Propagation-type CCL processing consists of initialization, neighborhood search, and label ID update. These three processes have pixel-by-pixel parallelism and are suitable for GPUs.

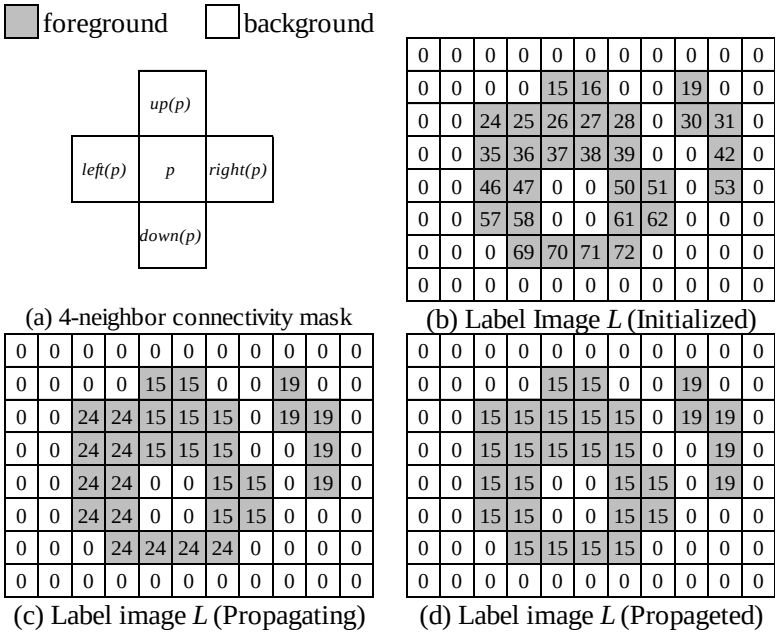

FIGURE 1: Propagation Type CCL.

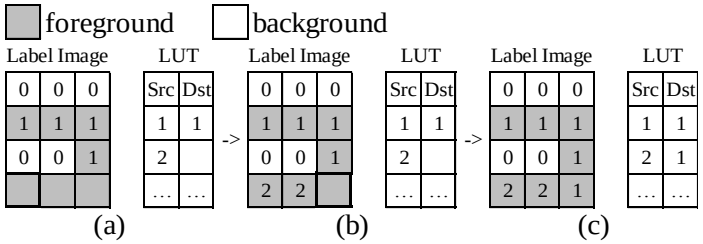

FIGURE 2: Example that cannot be connected by a single scan.

In the initialization process, the array index is stored in all pixels of the label image. The array index is calculated by *y x width + x*, where *(x, y)* is the pixel coordinate and width is the image width. On the outer periphery of the label image, background labels IDs are stored for simplicity.

The algorithm for CCL processing is shown in Algorithm 1. *PROPAGATE* is propagation processing and *SEARCH_AROUND* is neighborhood search. At lines 7 to 10 4 neighbors are searched, and at lines 11 to 15 the minimum label ID is store in the label image. *Up(), left(), down(), right()* at from line 7 to 10 are functions that return the relative neighborhood index of the argument *p*. Each correspondence is shown in FIGURE 1 (a). *Atomic* on lines 13 and 14 indicates an atomic operation. The atomic operation prohibits reading and writing to the variables of other threads until an atomic operation thread finishes the operation, when multiple threads operate on a shared variable such as a global variable at the same time. There are functions such as *min* and *max* in atomic operations. These functions return minimum or maximum in a variable on a single thread. This is described on lines 13 and 14. The atomic operation result is obtained by performing <- (assignment) within the scope of the atomic operation.

PROPAGATE is assumed to be executed in parallel, and the minimum label ID may be smaller after the neighborhood search than when 4 neighborhoods are referenced. So, by re-referencing

the label image with the searched minimum label ID, a small label ID that is farther away can be obtained in one scan, and the propagation speed is increased. By executing this propagation process multiple times, each connected component pixel group is replaced with the same unique label ID, and the connected component can be extracted. In the extracted label image, as shown in the FIGURE 1 (d), the smallest label ID (one-dimensional array index) among all the pixels in each connected component is propagated as a unique ID. Therefore, the maximum value of the label ID after propagation can be the number of pixels in the input image.

After line 24 is the processing flow for generating sequential label images. Labels are propagated by executing *PROPAGATE* multiple times. In order to minimize the number of executions of *PROPAGATE*, global synchronization is performed every time the propagation process for all pixels is completed.

Algorithm 1 CCL algorithm and conventional label continuation me

```
   Rquire:
      Image array B
   Ensure:
      Label image array L
 1 : function SEARCH_AROUND(t, n)
 2 :    if n != 0 and t > n then return n else retun t end if
 3 : end function
 4 : procedure PROPAGATE(L , p )
 5 :       g <- o <- L [p ]
 6 :    if o != 0 then
 7 :       g <- SEARCH_AROUND(g, L [up (p )])
 8 :       g <- SEARCH_AROUND(g, L [left (p )])
 9 :       g <- SEARCH_AROUND(g, L [down (p )])
10 :       g <- SEARCH_AROUND(g, L [right (p )])
11 :       g <- L [L [L [g]]]
12 :       if g != 0 then
13 :          atomic (L [o ] <- min (L [o ], g ))
14 :          atomic (L [p ] <- min (L [p ], g ))
15 :       end if
16 :    end if
17 : end procedure
18 : procedure UPDATE_LUT(L p , lastId )
19 :    if L [p ] != 0 and T [Lp ] = 0 then
20 :       lastId <- lastId + 1
21 :       T [Lp ] <- lastId
22 :    end if
23 : end procedure
24 : for p <- 1, IMAGE_SIZE do in parallel
25 :    if B [p ] = 0 then L [p ] <= 0 else L [p ] <- p end if
26 : end for
27 : for count <- 1, IMAX_NUM_LOOPS do
28 :    for p <- 1, IMAGE_SIZE do in parallel
29 :       PROPAGATE(L, p )
30 :    end for
31 : end for
32 : lastId <- 0
33 : for p <- 1, IMAGE_SIZE do
34 :    UPDATE_LUT(L [p ], lastId )
35 : end for
```

2.1.2. Related Works on CCL

The CCL algorithm has been researched for a long time [1]. Several methods for ensuring parallelism have been proposed for the problems of raster algorithms that lack simple parallelism.

Raster Type

Suzuki et al. proposed a four-time scanning-type CCL algorithm whose processing time was linearly proportional to the image size without depending on the geometric shape, by performing forward search from the upper left and backward search from the lower right [9]. Double-scan CCL algorithm proposed by Wu et al. records the link information between multiple provisional labels pasted during a first scan, and changes the global label link relationship by Union-find [10]. The process is completed by writing back the label using the connection relation information during the last second scan. The two-scan CCL proposed by He et al. reduces the memory access by evaluating the identity of the label for each connected component unit for each raster, instead of evaluating for each pixel at the time of scanning [11].

Propagation Type

[12] proposes a parallel algorithm of CCL, and achieves a speed increase of about 20 times on a 24-core CPU. [13] proposes two algorithms for GPGPU. One is a technique that applies reduction processing. The other is a combination of a general label connectivity evaluation method and a neighborhood search method. In [6], the speed is improved by dividing the input image and transferring to a high-speed on-chip memory, and then propagating labels on it. In [7][14] [15], the propagation speed of the label ID is improved by searching the minimum label ID again using the label ID searched after the peripheral pixel search considering the characteristics of asynchronous processing.

2.1.3. Application of CCL

CCL is mainly used for object detection and diagnosis using binary images generated from two-dimensional features. Application examples are product defect diagnosis on production lines, pest diagnosis of crops, yield estimation, medical image diagnosis using optical images and CT images, and object detection.

[16] aims to improve production efficiency by detecting defective products by strain inspection early in the manufacturing process of rearview mirrors. The inspection proposed here is realized by irradiating a glass surface with an equally-spaced circular pattern, capturing the image, extracting each circular pattern from the captured image, and using the distance relationship between the circular patterns. CCL process is used when this circular pattern is detected. [17] has realized automation of surface inspection of rolled steel products. The research on automation of this surface inspection has been devised for a long time, and it was initially used for rough inspection by laser, and now, research is being made toward the realization of relatively high precision inspection using CCD. In this document, the hue and the reflection of light are specified from the image of the product surface, the defective area is specified from the color difference and the luminance, and the area is extracted by CLL.

[3] proposes an inspection system that automatically extracts concrete cracks from captured road surface images. Reflection of outside light is extremely limited, since the cracked area becomes a groove. Using this phenomenon, the image is binarized using the characteristic luminance and color difference that appears when the image is taken, and the crack region is detected by CCL. [18] has proposed a lesion diagnosis system using optical images for the purpose of disease diagnosis of crops. Diseases of agricultural crops have characteristic patterns and color differences in the leaves, so disease diagnosis is performed by extracting those features. In addition, the severity of the disease is quantitatively measured by investigating the ratio of the lesion area to the entire leaf area. Information on leaf area or lesion area extracted by CCL is used for both disease diagnosis and severity calculation.

Medical diagnosis using medical images has been actively studied for a long time. In [19], the abnormality diagnosis of red blood cells is automated, and the image captured by the microscope

is used to binarize the color features and the platelets are extracted by CCL. [2] automates the detection of platelets and the detection of malaria parasites in the blood by image processing. In [20][21][22][23][24][25], by extracting organs, blood vessels, cancer, and tumor areas using single or multiple slices, The 3D CG reconstruction of organ images, the extraction of lesion areas due to disease, and disease estimation diagnosis are realized. In both cases, connected components are extracted from CT images to extract organs, blood vessels, and lesion areas, and features such as shape and hue are used for each connected component.

In object detection, [4][26][27][28][29][30] proposes face detection using color features, and [28][31][32] estimates the gaze area and face area by analyzing the positional relationship nd movement of the eyeball from the captured image. In object detection, feature image is generated, and CCL is used to extract a face region block. In [33], in order to suppress the disturbance caused by glasses in face recognition, the glasses area is automatically extracted and removed from the face image. In object detection, the feature image is generated in all cases and CCL is used to extract the face area blocks.

[7][13] has developed and verified a new AR marker that can be recognized at high speed while maintaining the recognition accuracy of the existing AR marker or higher. CCL is used to extract the proposed marker pattern from the image, and it is accelerated by GPGPU. [34][35][36] attempted to recognize characters and graphs from scanned sentences or cartoon images. Connected components are extracted from the binary image generated by colors and image filters, and characters and graphs are extracted as one component by using the area of each component and the correlation with the neighboring components of the component.

2.1.4. Summary and Issues of CCL

As explained in section 2.1.3, CCL is used in various fields of applications. CCL is used to calculate the area of each component and the distance and number of components. However, the label ID of the final label image is discontinuous as in FIGURE 1 (d), and the maximum value of ID can be the total number of pixels in the input image. In the implementation that counts the area by array using the label ID of each component as a key, memory access is discontinuous, it is not suitable for CPU speed-up mechanism, and resource utilization efficiency is poor. Furthermore, threshold processing using the calculated area is performed in units of labels, but processing for all pixels is required since all label IDs must be processed. In CCL, it is extremely important to perform the above-mentioned label re-sticking process at high speed. This is because it is used in applications that require extremely high real-time properties such as product inspection, medical diagnosis using images, and object recognition of faces and markers.

Machine learning methods such as SVM, which was developed in the 2000s, and deep learning, which has been attracting a lot of attention since the 2010s, have extremely high accuracy. However, it requires a learning phase over several tens of hours by creating learning datasets with abundant volumes and using high-performance computing resources. In addition, relatively high computational resources are often required in also the detection and recognition phases of actual operations. Therefore, it cannot always be stated that these machine learning methods are suitable for high-speed inspection systems using embedded devices. Considering the trade-off between the inspection speed and the processing speed, there is a high need for inspection systems based on manual feature extraction and region extraction.

2.2. HPF using Background Connected Components

Here, the HFP is explained using the FIGURE 3 for the diagnosis of leaf disease by image processing. (a) is an input image, and (b) is a binary image obtained by thresholding (a) based on leaf color information. The HFP is used to complement the leaf area that could not be extracted from the input image by threshold processing. The threshold is experimentally set using the normal green feature of the leaf, but there is a disease region in the leaf of (a), and the color is changing. Therefore, the lesion area cannot be extracted as a leaf area as shown in (b). As shown in (b), the lesion area inside the leaf area is determined as background areas such as holes. Therefore, the leaf area is complemented by using a HFP that makes the background area

foreground. The result is (c). The procedure of this HFP is realized by extracting the connected components of the background region, generating the area histogram of the background components, and then foregrounding the small background components.

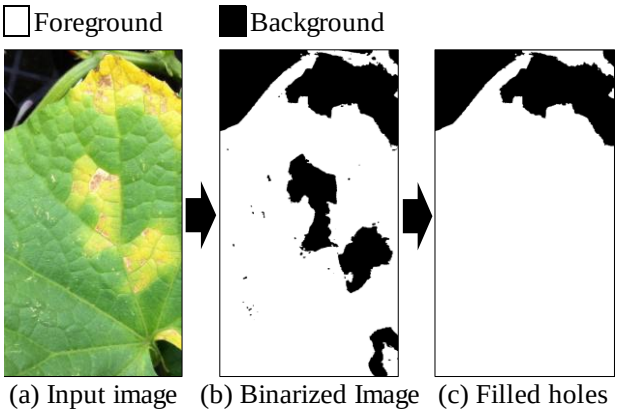

FIGURE 3: HFP on Binary Image.

2.2.1. Application of HFP

One of the practical uses of extracting objects and noise areas in an image is processing to fill non-object areas (holes) existing in the object area [2][4][5][18]. In [2], when binarization using color information is performed to extract the blood cell system existing in the blood, red blood cells are often imaged in a donut shape, and the central part is missing and extracted. Therefore, the HFP is used to extract the central part of the donut shape as the same object. [4] is a simple face detection process based on an image containing a face area photographed under limited conditions. The HFP is applied to complement the face area that could not be extracted by the hue information. [5] is a study of the theory of bubbles rising in a liquid. Observing and reconstructing the actual behavior in the natural world is extremely important in clarifying the principle. Using an experimental device that generates bubbles in the liquid, the behavior of the bubbles is photographed and they understand the behavior by tracking the position. The purpose of this research is to automate the behavior analysis. They have proposed a method that extracts the outline of bubbles from a photographed image and extracts the entire bubble as a single object by HFP. After that, they analyses by matching using a bubble shape model. [18] diagnoses disease using leaf images as described in section 2.2. The HFP is used to integrate the missing area into the leaf area since the lesion area of the leaf is discolored.

By connecting non-object connected components with a small area to the object connected components by the HFP, it is possible to extract the object region from which noise has been removed. Processing methods such as histogram calculation are used to calculate the area of non-object connected components.

2.3. GPGPU

In recent years, the GPU, which is a processor on a graphic board, has grown significantly, and now it has reached the theoretical performance more than 10 times the CPU. The GPU is a processor optimized for graphics applications such as 3D modeling and computer graphics, but general-purpose scientific and technical computations can be processed at high speed if the algorithm is suitable for this GPU architecture. For this reason, research and development of computation methods suitable for this GPU has been actively pursued in recent years and is called GPGPU (General Purpose Graphics Processing Units). Among these, application results have been reported in various fields such as image processing, weather forecasting, and encryption processing. However, in order to maximize the original performance of the processor in application execution, there is a problem that it is necessary to understand the GPU structure and apply an execution control method suitable for it.

2.3.1. Mobile GPGPU Environment

The performance of so-called SoC (System-on-a-Chip) which is a processor for mobile terminals such as smartphones and tablet PCs, has made remarkable progress. In March 2017, NVIDIA announced Tegra X2, an SoC that integrates a CPU and GPU [8]. Although the TX2 is a very small board of 50x87mm square and business card size as shown in FIGURE 4, it is equipped with 256 CUDA cores used in also desktop systems and supports general-purpose computing technology. NVIDIA Jetson TX2 development kit equipped with Tegra X2 was released. This is a small board size of 13 inches square and a low price of $599. This TX2 meets the constraints such as size, weight and power consumption when loading on a drone and is very suitable for our system.

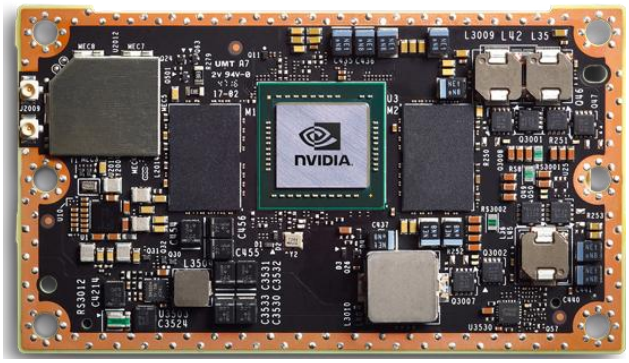

FIGURE 4: NVIDIA Tegra X2 Board.

3. PROPOSED METHOD

In this section, we propose a rewriting algorithm for continuous label ID for GPGPU, a hole-filling algorithm using connected components extracted simultaneously foreground and background, and a reduction and speedup of the calculation.

3.1. Label Rewriting Algorithm

In the application implementation of this research, the area of the connected component is calculated. This process is so-called histogram calculation. In a processing system that does not have a dictionary data structure, it is calculated by incrementing the array element for recording using the label ID as a key. However, after CCL, the label ID is not continuous from 1 as in the FIGURE 1 (d). So, Cache hit rate is significantly reduced since the array is accessed using the non-continuous label ID as a key. In addition, there are many unused memory areas and the memory utilization efficiency is low. Furthermore, when this processing is executed in parallel on the GPU, a memory area that is the number of parallel executions multiplied by the number of pixels is required, which is unrealistic.

One solution to this problem is to rewrite non-contiguous label IDs to continuous label IDs starting from 1. The rewriting process is shown on lines 18 to 23 of Algorithm 1. *T* is a LUT for recording the correspondence between non-continuous labels and continuous labels for each component. *T* is a one-dimensional array with elements for the number of pixels, the indices are non-continuous labels, and the element values are continuous label IDs. The variable *lastId* holds the label ID that was last acquired. By incrementing this variable, a new continuous ID can be acquired and *T* can be updated. Since the variable *lastid* cannot be incremented simultaneously, the rewriting process cannot be simply parallelized. Therefore, when updating *T* on a GPU, almost all cores become idle, processing time is long, execution efficiency is extremely low, and it is not suitable for GPGPU. So, we propose a rewriting algorithm based on IF statement control for continuous labels with pixel-by-pixel parallelism.

The algorithm is shown in lines 16 to 22 of Algorithm 2. The ID of the label image extracted from connected components used in this study is the array index. Also, since each thread is

responsible for processing one pixel, the thread always has a unique ID, which is the same coordinate system as the array index of the label image. Each component of the final label image extracted from the connected component has a minimum array index written as a label ID, and a thread having a thread ID that matches the minimum label ID is uniquely determined. So, the thread whose thread ID and label ID match becomes the representative thread for that component. After that, the representative thread acquires a unique continuous label ID, records the relationship between the non-continuous label ID and the continuous ID in the LUT, and uses the LUT to rewrite the label image with the continuous label ID.

Algorithm 2 Simultaneous foreground and background CCL method and label continuation method

```
Rquire:
    Image array B
Ensure:
    Label image array L look-up table array T, label to array index look-up table
    LabelToIdx, last label ID LastId
 1 : function SEARCH_AROUND2(bt, t, bn, n)
 2 :     if bt xor bn != 0 and t > n then return n else retun t end if
 3 : end function
 4 : procedure PROPAGATE2(B, L, p)
 5 :     g <- o <- L[p]
 6 :     g <- SEARCH_AROUND2(B[p], g, B[up(p], L[up(p)])
 7 :     g <- SEARCH_AROUND2(B[p], g, B[left(p], L[left(p)])
 8 :     g <- SEARCH_AROUND2(B[p], g, B[down(p], L[down(p)])
 9 :     g <- SEARCH_AROUND2(B[p], g, B[right(p], L[right(p)])
10 :     g <- L[L[L[L[g]]]]
11 :     if g != 0 then
12 :         atomic(L[o] <- min(L[o], g))
13 :         atomic(L[p] <- min(L[p], g))
14 :     end if
15 : end procedure
16 : procedure UPDATE_LUT_BY_SAME_PIXELVALUE(L, T, LabelToIdx, p,
17 :     if T[L[p]] = p then
18 :         atomic(u <- ++lastId)
19 :         T[L[p]] <- max(T[L[p]], u)
20          LabelToIdx[L[p]] <- u
21 :     end if
22 : end procedure
23 : for p <- 1, IMAGE_SIZE do in parallel
24 :     L[p] <- p
25       T[p] <- 0
26 : end for
27 : for count <- 1, IMAX_NUM_LOOPS do
28 :     for p <- 1, IMAGE_SIZE do in parallel
29 :         PROPAGATE2(B, L, p)
30 :     end for
31 : end for
32 : lastId <- 0
33 : for p <- 1, IMAGE_SIZE do in parallel
34 :     UPDATE_LUT_BY_SAME_PIXELVALUE(L, T, p, lastId)
35 : end for
```

3.2. Foreground and Background Simultaneous CCL

The outline is the same as the algorithm described in the section 2.1. Only the parts that differ greatly are extracted and explained using Algorithm 2.

In the initialization process, the connected components are extracted at the same time for the foreground and background, so a one-dimensional array index is stored for all pixels except for 1px around the label image. There are two main differences between the foreground and background simultaneous propagation processing and the foreground or background only propagation processing. In the vicinity search of Algorithm 1, if the target pixel is the background, the connection judgment of the surrounding pixels is not performed. The first difference is that the foreground and background are extracted simultaneously in this process, so the connection judgment is performed regardless of the target pixel value. The second difference is that this process judges whether the values of the binary image corresponding to the target pixel and the surrounding pixels are the same. For this judgment, XOR is used for speeding up, and the property that it always becomes 0 when the input values are the same is used.

Finally, the process of rewriting to continuous label ID is shown on lines 16 to 26. The LUT is checked for all pixels, since both the foreground and background are rewritten simultaneously. If the label of the LUT is the initial value, the representative thread of each component acquires the new label ID and stores it.

As described in section 3.1, label IDs of the label image are rewritten to the serial number. However, even if the label image is rewritten to the serial number label at the end of this CCL process, it is necessary to rewrite the serial number label ID again in order to integrate the foreground components by the HFP. So, it is a slow and useless calculation. Therefore, in the fore-background simultaneous CCL used for the HFP, only generation of the LUT that records the relationship between the non-continuous label ID and the continuous ID is performed, and the label is rewritten after the HFP. The continuous label ID is rewritten to a unique ID of the same set regardless of the foreground / background, since the foreground / background can be judged from binary images.

3.3. HFP using Foreground and Background Simultaneous CCL

In this section, we propose a HFP using connected components that have been extracted simultaneously the foreground and background proposed in section 3.2. We explain using a FIGURE 5 which shows briefly the relationship between the foreground and background labels. (a) is a label image that records the connected components extracted simultaneously with the foreground and background at the section 3.2, and this is the initial value for the filling process. The index prefixed with L in the figure is the label ID of each component, and the magnitude relationship of the index part matches the nature of the original label ID. In this HFP, the connected components of the foreground and background have already been extracted. The components to be filled are background components, but its neighboring components are always foreground components, so the foreground component label IDs are integrated after foregrounding the background components.

Step1. Area calculation of Foreground and Background Components and Extraction of Small Components

A histogram of the number of pixels for each foreground / background component is generated. Unlike the section 2.2, a histogram for both foreground and background components is generated. This is because, at the end of the section 3.2, a new continuous label ID is generated for the foreground and background, and it corresponds to it. Label IDs of the label image are rewritten with the continuous label IDs of this LUT, and a histogram is generated with the continuous label IDs. It is possible to generate a histogram only for the background, but in this case, background label judgment is required using a conditional expression after referring to the binary image. Conditional expressions can cause stalls in multistage pipelines of computing units, and SIMD operations that perform the same processing on multiple data cannot be applied, which can cause speed performance degradation. Therefore, a histogram is generated for both the foreground and background.

Next, if each component area is less than or equal to the threshold value, True is stored in the integration flag of the component. In (b), "Filling" are written as background components to fill the hole, and these are {L3, L5}.

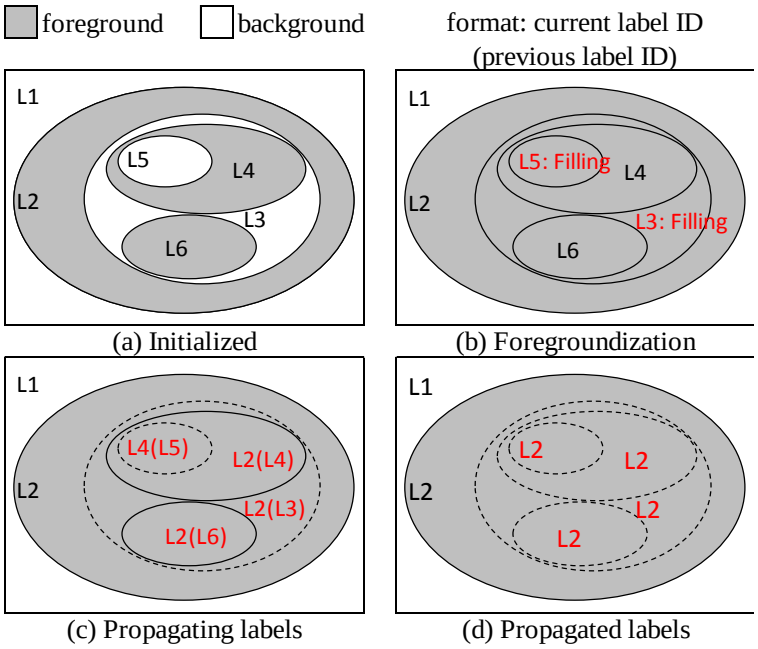

FIGURE 5: Foregroundization of Small Background Components.

Step2. Foregroundization of Small Background Components

Next, the binary image that records the foreground / background information of each pixel is rewritten. The integrated judgment result of each label and the binary image are of the same type, and the integrated judgment result of each label is added to the binary image as a foreground pixel by bit operation.

Step3. Integrated Search for Foreground and Background Labels

In Step 2, the L3 and L5 labels in (b) have become foreground from the background, so the labels L2-L6 all become the same connected component. However, each component has a different label ID, and it is necessary to integrate different label IDs of these same components into the same label ID. Here, we propose a method for integrating different label IDs of the same component. The algorithm is shown in Algorithm 3. It is designed to produce correct results even if executed in parallel. *MERGING_FOREGROUND* searches for foreground components for 4 neighborhoods, and *SMF* is the search algorithm. If the component is foreground and the neighborhood label ID is smaller than the label ID, the neighborhood label ID is used as the label ID. LUT is used to refer to and rewrite label IDs. In this algorithm, only foreground label IDs are propagated, and background components become the initial value 0 (background label ID) of the LUT. (c) shows the propagation process, and (d) shows the integrated LUT that completed the propagation. In (c), the components that propagated the label IDs describe the label IDs before and after propagation.

Step4. Generation of New Continuous Label ID Correspondence Information

Since label IDs propagated in Step 3 are array indexes, these are non-consecutive numbers. Therefore, a LUT for rewriting the foreground label IDs after integration into continuous label IDs is generated. This algorithm is shown on lines 15-24. *T* records the relationship between label IDs at the time of CCL and the minimum foreground label IDs integrated by foreground, and *T'* records the correspondence between the integrated label IDs and the new continuous label IDs. Note that unlike the acquisition of continuous label IDs in the section 3.1, it is sufficient in this

process to perform label update checks for the total number of labels in foreground / background CCL.

Algorithm 3 HFP using Foreground and Background Simultaneous

```
Rquire:
   Image array B, look-up table array T, label to arrayindex look-up
   table LabelToIdx, last label ID lastId
Ensure:
   Label image array L
1 : function SMF(T, t, bn, n)
2 :    if bn = 1 and T[t] > T[n] then return n else retun t end if
3 : end function
4 : procedure MERGING_FOREGROUND(B, L, T, p)
5 :    if B[p] = 1 then
6 :       g <- L[p]
7 :       g <- SMF(T, g, B[up(p)], L[up(p)])
8 :       g <- SMF(T, g, B[left(p)], L[left(p)])
9 :       g -> L[L[g]]
10 :      if T[L[p]] != g then
11 :         atomic(T[L[p]] <- min(T[L[p]], T[g]))
12 :      end if
13 :   end if
14 : end procedure
15 : procedure UPDATE_LUT_FOR FILLING HOLES(T, T', I, lastId)
16 :   if T'[T[I]] = 0 then
17 :      atomic(v <- --T'[T[I]])
18 :      syncthreads()
19        if v = -1 then
20           atomic(lastId <- i+1)
21           T'[T[I]] <- lastId
22        end if
23 :   end if
24 : end procedure
25 : for count <- 1, IMAX_NUM_LOOPS do
26 :   for I <- 1, lastId do in parallel
27 :      MERING_FOREGROUND(B, L, T, labelToIdx[I])
28 :   end for
29 : end for
30 : lastId' <- 0
31 : for p <- 1, IMAGE_SIZE do in parallel
32 :   UPDATE_LUT_FOR_FILLING_HOLES(T, T', I, lastId')
33 : end for
```

3.4. Optimization HFP using Foreground and Background Simultaneous CCL

In the section 3.3, the HFP for GPGPU is explained. In the HFP, background components that became the foreground are always connected to the surrounding foreground components. As a result, multiple foreground components may be connected to surrounding foreground components. The computational order of this search process can be reduced from the total number of input image pixels to the number of foreground and background labels. This reduction method is explained in this section.

As explained in the section 3.3, foreground component integration is achieved by searching for connectivity to the neighborhood of all pixels in the image and propagating a smaller foreground

label ID. As propagating label IDs, discontinuous label IDs used when extracting connected components are used. These label IDs are array indexes, and the label ID of each component is the smallest array index of each component, that is, the array index existing at the upper left of the same component. It is guaranteed that the label IDs of components that exist above or to the left of a component are always smaller than the label ID of that component, and conversely, the label IDs of components that exist below or to the right are greater than the label ID of that component.

All the pixels were searched for the connectivity check of the component. As mentioned above, the label ID of a certain component is the smallest pixel index existing at the upper left of the component. In other words, by referring to the label image by each label ID (pixel index) and searching the pixels above and to the left of the pixels, the connectivity of all the components can always be checked. As a result, in the connectivity check process, all the pixels have been searched so far, but it is sufficient to search all the number of labels Ln times in the sequential processing. When executed in parallel on the GPU, it is possible to link all the labels with a shallow component nesting by searching all the labels about twice, but for an image with a deep component nesting, it is necessary to search all the labels about 8 times or more. The foreground component integration algorithm using these properties is shown on lines 25 to 29 of Algorithm 3.

4. EXPERIMENTAL RESULTS

In this section, we verify and evaluate the speedup rate and computational efficiency of the HFP by the background and background simultaneous CCL using the conventional method and the proposed method. To verify GPGPU implementation, NVIDIA's Jetson TX2 was used, so it was implemented with CUDA (Compute Unified Device Architecture), a development environment provided by the company.

4.1. Experimental Environment

As the execution environment, we used the above-mentioned CPU and GPU on TX2. TABLE 1 shows the experimental environment.

Spec	Jetson TX2
CPU	Cortex-A57@2GHx4 Cores
# GFLOPS	16.0
GPU	Pascal@1.3GHz 256 Cores
# GFLOPS	665.6
OS	Ubuntu 16.04 LST

TABLE 1: Experiment Environment.

Image Size	Laster Type			Propagation Type			Rtio		
	Leaf	Circle	Rect.	Leaf	Circle	Rect.	Leaf	Circle	Rect.
480 x 270	1,506	1,588	1,539	1,752	2,326	1,692	1.16	1.47	1.10
960 x 540	3,754	4,833	2,740	6,735	8,176	6,291	1.79	1.69	2.30
1,920 x 1,080	10,732	11,874	6,379	25,984	32,587	24,506	2.42	2.74	3.84
3,840 x 2,160	36,425	44,542	21,243	104,108	129,558	97.225	2.86	2.91	4.58
7,680 x 4,320	134,786	200,298	75,975	401,985	540.460	492.963	2.98	2.70	6.49

TABLE 2: Raster type and propagation type processing time (microsecond) and its ratio in CPU.

The two CCL methods described in the section 2.1 are implemented for the CPU, and the result is shown in TABLE 2. Propagation-type CCL was about 1.1-6.5 times slower than the raster type. For this reason, on the CPU, raster-type CCL was adopted for HFP.

4.2. Propagation of Foreground Component Labels and its Verification

In our proposed method, the HFP is realized by integrating the foreground components. This integration process is performed by propagating the minimum label ID in the connected component as described in the section 3.3. However, in the image with nested foreground and background components such as FIGURE 6, when the outer-most or inter-most component ID is the smallest, it is need to propagate N-1 times to integrate all connected foreground components, where N is the number of nested foreground components. In sequential execution, it will be a raster scan from the upper left to the lower right of the image, so propagation is always completed with one scan. In the worst case for propagation, if all pixels are processed in parallel, all pixel values are acquired and the neighborhood search and minimum label ID are stored simultaneously. So, the label ID that can be propagated in one scan is one neighborhood. For this reason, it is necessary to scan all pixels N-1 times to complete the propagation process. However, by limiting the number of execution threads during this operation, the timing of scanning each pixel is shifted, and there is a high possibility that a small label ID will propagate more quickly. In this experiment, the number of scanning processes described above was set to 1.

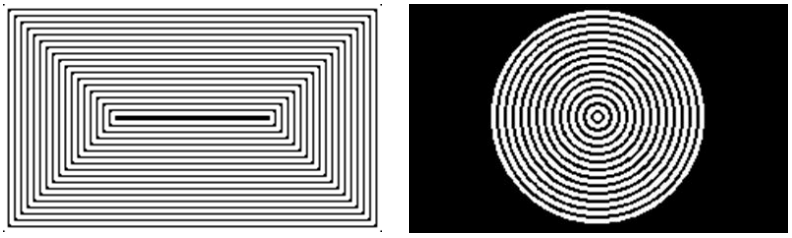

FIGURE 6: Nested Images.

In this experiment, 5 types of leaf images and 2 types of artificial images were prepared and converted to 5 image sizes, respectively and we used them. The HFP was performed 1,000 times for this data set. It was confirmed that all connected foreground components of the label image generated at each execution were successfully connected.

4.3. Speedup of Rewriting Non-sequential Label IDs to Sequential Numbers

We verified a parallel algorithm for rewriting non-continuous label IDs into continuous IDs described in the section 3.1. The conventional sequential processing was implemented and executed on one core of the GPU and CPU. However, the GPU is about 100 times slower than other methods, so the evaluation of the results is omitted. Each processing time and the acceleration rate are shown in TABLE 3. And, the graph that visualized the acceleration rate is shown in FIGURE 7.

Image Size	Proposed on GPU			Conventional on CPU			Conv. / Prop.		
	Leaf	Circle	Rect.	Leaf	Circle	Rect.	Leaf	Circle	Rect.
480 x 270	170	172	172	650	640	664	3.83	3.72	3.87
960 x 540	552	585	585	2,231	2,190	2,215	4.04	3.74	3.78
1,920 x 1,080	1,959	2,100	2,105	27,047	27,162	27,317	13.81	12.94	12.97
3,840 x 2,160	7,806	8,582	8,484	105,210	106,228	106,468	13.48	12.38	12.55
7,680 x 4,320	31,527	35,471	34,304	417,027	421,328	417,174	13.23	11.88	12.16

TABLE 3: Processing time (microseconds) and acceleration rate of the label rewriting.

Compared with the conventional method, the proposed method using equivalence judgment is about 3.7-13.8 times faster, and on average about 9.2 times faster. For the proposed method, execution time and image size are linearly proportional. In the conventional method, as shown in FIGURE 7, there is a difference in processing time when the image size is 960x540 or less and otherwise. This is probably because the memory usage of the label image is small, about 500 KB, and the cache hit rate is high.

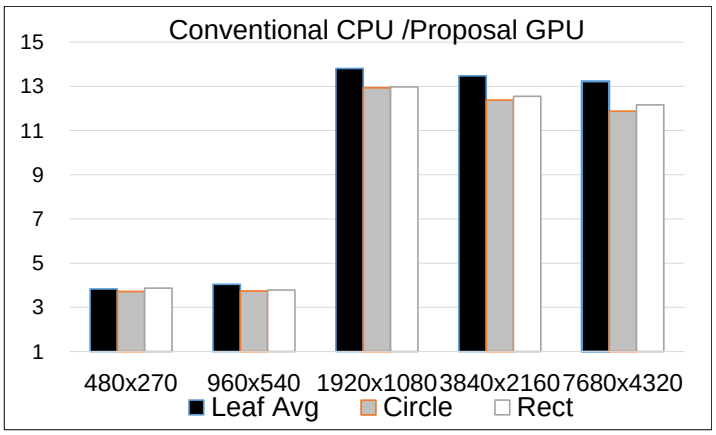

FIGURE 7: Acceleration rate and processing time increase rate in label rewriting process.

4.4. Speedup of Histogram Generation due to Label Rewriting

Here, we evaluate the speedup of histogram generation due to label rewriting. By rewriting to a continuous label, the cache hit rate is improved, but it is verified whether the processing time is shortened including the rewriting time. TABLE 4 shows histogram generation time, label rewriting time and speed up rate on TX2 and NVIDIA GeforceGTX970M for the deskside PC. In the table, "Hist." is the histogram generation time, and "Total" is the time including label rewriting. "Conv." is a conventional method that does not rewrite labels, and only the time for the histogram generation is "Total".

Type	Image Size	on TX2					on Deskside PC (GTX970M)				
		Proposed		Conv.	Speedup		Proposed		Conv.	Speedup	
		Hist.	Total	Hist.	Hist.	Total	Hist.	Total	Hist.	Hist.	Total
Leaf Avg.	480 x 270	305	474	330	1.08	0.70	152	192	187	1.23	0.98
	960 x 540	734	1,292	804	1.10	0.62	368	480	586	1.59	1.22
	1,920 x 1,080	2,375	4,367	2,595	1.09	0.59	960	1,351	2,093	2.18	1.55
	3,840 x 2,160	9.077	17,112	12,458	1.37	0.73	3,060	4,467	6,804	2.22	1.52
	7,680 x 4,320	36,044	68,628	42,203	1.17	0.62	11,980	17,553	27,435	2.29	1.56
Circle	480 x 270	296	468	333	1.12	0.71	149	190	196	1.32	1.04
	960 x 540	691	1,276	772	1.12	0.61	355	472	638	1.80	1.35
	1,920 x 1,080	2,161	4,261	2,657	1.23	0.62	898	1,308	2,037	2.27	1.56
	3,840 x 2,160	8,145	16,727	12,410	1.52	0.74	2,781	4,322	7,355	2.65	1.70
	7,680 x 4,320	32,071	67,543	40,898	1.28	0.61	10,767	17,106	28,893	2.68	1.69
Rectangle	480 x 270	301	472	320	1.07	0.68	149	189	197	1.32	1.04
	960 x 540	707	1,292	872	1.23	0.68	359	474	636	1.77	1.34
	1,920 x 1,080	2,222	4,328	3,000	1.35	0.69	915	1,322	2,009	2.29	1.59
	3,840 x 2,160	8,362	16,845	11,550	1.38	0.69	2,855	4,326	7,306	2.56	1.69
	7,680 x 4,320	32,895	67,198	46,927	1.42	0.70	11,021	17,009	29,345	2.66	1.73

TABLE 4: Histogram generation time (microseconds) and speedup rate due to label rewriting.

The histogram generation is explained. In the conventional method, the size of the histogram bin corresponds to the resolution of the input image. For this reason, a high-speed histogram generation method using shared memory cannot be used unless the image size is small. Instead, the histogram is generated by atomic operation to the histogram bin stored in the global memory. On the other hand, in the proposed method, the size of the histogram bin is reduced to the

number of connected components by rewriting the label to continuous IDs, so that a high-speed shared memory can be used.

In the execution results on TX2, the proposed method is slower. On the other hand, the proposed method is faster on the GeforceGTX970M. The histogram generation time ratio of the conventional / proposed method on TX2 is about 1.1-1.5 times, while that on the GTX970M is 1.2-2.7 times. This difference can be attributed to an increase in the number of locks in atomic operations due to an increase in the number of concurrently executing threads. TX2 has 256 CUDA cores, while GTX970M has 1,280 cores. On the GTX970M with a large number of cores, effect of using high-speed shared memory for histogram generation in the proposed method is large. As a result, the increase in the number of cores increases the access probability to the same element, and it is thought that the operation delay is caused by the lock of the atomic operation.

4.5. Whole Execution Time of HFP

TABLE 5 shows processing time and speedup rate of HFP. The number of pixels in each image is four times larger, and it can be seen that the processing time is roughly proportional to the number of pixels. The speedup rate of the proposed method compared with the conventional method of GPU execution is 1.19-1.27 times for leaf images, which are practical data sets. For artificial images, it is 1.14-1.22 times for Circle and 1.02-1.04 times for Rectangle. In Rectangle, the proposed method is slower, but the performance of the proposed method is higher in other images. The speedup rate of the proposed method compared with the conventional method of CPU execution is 1.40-2.46 times for leaf images, 1.22-2.16 times for Circle and 1.12-1.97 times for Rectangle. Again, some of the proposed methods are slower in Rectangle, but the performance of the proposed method is higher in other images.

Image Type	Image Size	Proposed on GPU	Conventional on GPU	Conventional on CPU	Speedup vs. GPU	Speedup vs. CPU
Leaf Avg.	480 x 270	1,521	1,808	3,741	1.19	2.46
	960 x 540	5,039	6,807	10,242	1.13	2.03
	1,920 x 1,080	18,604	23,507	30,336	1.26	1.63
	3,840 x 2,160	72,423	88,363	104,895	1.22	1.45
	7,680 x 4,320	280,091	355,982	392,934	1.27	1.40
Circle	480 x 270	1,603	1,849	3,461	1.15	2.16
	960 x 540	5,661	6,879	11,023	1.22	1.95
	1,920 x 1,080	20,261	23,815	26,858	1.18	1.22
	3,840 x 2,160	81,352	92,764	99,272	1.14	1.22
	7,680 x 4,320	328,415	375,714	412,114	1.14	1.26
Rectangle	480 x 270	1,994	2,031	3,933	1.02	1.97
	960 x 540	7,791	7,923	8,730	1.02	1.12
	1,920 x 1,080	27,728	28,702	23,968	1.04	0.86
	3,840 x 2,160	126,694	116,933	85,397	0.92	0.67
	7,680 x 4,320	547,248	467,065	327,606	0.85	0.60

TABLE 5: Processing time (microseconds) and speedup rate of HFP.

TABLE 6 shows the processing time of each kernel in each image of the proposed method and the conventional method. Only the results for images with a resolution equivalent to Full HD (1,920 x 1,080) are shown in this table, but it has been confirmed that the trend of processing time variation described below is the same for all resolution images. The leaf images are extracted by two different features. “Number of Pixels” indicates the number of pixels in the background connected component (Back-ground) and foreground connected component (Fore-ground) of each image. “FB-CCL”, “HFP” corresponds to foreground and background

simultaneous connected component labeling and hole filling process, respectively. “B-CCL”, “HFP”, “F-CCL” corresponds to background connected component labeling, hole filling process, and foreground connected component labeling, respectively.

Image	Number of Pixels		Processing Time				
	Back-ground	Fore-ground	Proposed		Conventional		
			FB-CCL	HFP	B-CCL	HFP	F-CCL
Leaf A	662,419	1,411,181	14,401	4,454	9,388	2,393	12,114
Leaf B	1,113,069	960,531	14,199	4,141	10,914	2,389	10,409
Circle	1,533,462	540,138	16,088	4,173	12,579	2,161	9,075
Rectangle	1,037,880	1,035,720	21,082	4,646	14,510	2,222	11,970

TABLE 6: Each kernel processing time (microseconds).

There is a processing time difference of about 10-20% between leaf images and artificial images. In the case of the conventional method, the difference is due to the connected component labeling time. Binary images, which are the input of "B-CCL" and "F-CCL" for leaf images, have a large foreground component area, so the connected component extraction time was increased. In the case of the artificial image Rectangle, the connected component labeling time also increased as in the case of the leaf image because the area of the foreground component of the input binary image of "F-CCL" was large.

The ratio of the HFP time of the proposed method to the total processing time is much larger than that of the conventional method, and the execution time is about 1.7 to 3.0 times that of the conventional method. This is due to the foreground integration process which has a large amount of computation in the proposed method.

The conventional method used for comparison in Sections 4.3, 4.4 is the method used in many of the various systems shown in Section 2.1.3. Similarly, the conventional method used in Section 4.5 is the method used in many of the various systems shown in Section 2.2.1. Therefore, the results in Sections 4.3, 4.4, and 4.5 show that the proposed method is superior to other systems.

4.6. Applying to Object Detection

We evaluate the application of our proposed HFP method using simultaneous CCL of foreground and background. The application is a lightweight object (face) detection using CCL.

Object detection is a method to estimate the coordinates and area of the target object from the input image. This technique is often used as preprocessing for object recognition, and can improve recognition accuracy by limiting the image which is input to the classifier to the object region to be recognized. As a general and specific example of object detection, we explain using facial detection. There are four processing procedures: a) preprocessing of the input image, b) feature extraction, c) entire region search using the coincidence degree, and d) object region estimation. B) has a significant effect on the facial detection accuracy. A) includes noise reduction of images and correction of camera lens characteristics. C) searches the entire area for face position, using the likelihood of a face-like or human-skin-like in each pixel or region defined in b). Therefore, this process is computationally expensive and takes a huge amount of time.

4.6.1. General Object Detection Method

This section describes the commonly used HOG features and SVM-based object detection. HOG (Histogram of Oriented Gradients) [37] calculates a histogram of the luminance gradient and uses the histogram as a feature value. It is often used as a feature of objects with distinctive edges such as cars, human bodies, and faces. In object detection using HOG, a detection area called ROI (Region of Interesting) is defined, and the degree of coincidence with the detection target object is measured using images in the ROI area. Using this, it evaluates and estimates which position of the input image is close to the feature of the object to be detected for each ROI. The parameters that define the ROI are often defined by the position (x, y) in the image, the ROI size

(w, h), and the ROI scale. Since the face size in the image space differs depending on the distance between the camera and the imaged object and individual differences, the ROI scale is defined to cope with this change. As shown in FIG. 1, the specific extraction of HOG features is performed by a) dividing the ROI into cells, b) calculating the luminance gradient for each cell, and c) generating a luminance gradient histogram.

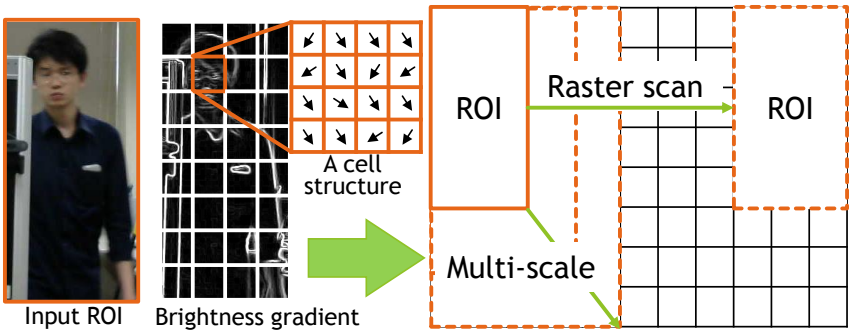

FIGURE 8: Overview of HOG processing and raster scan.

SVM (Support Vector Machine) [38] is a kind of machine learning method that can be applied to classification and regression problems. In face detection, it is determined whether the ROI is a face or not as a binary classification problem. The advantage of SVM is that it increases the generalization ability by maximizing the margin in the feature space.

4.6.2. Lightweight Object Detection Method using CCL

In this section, we propose a high-speed face candidate extraction process using a skin color filter. FIGURE 9 shows the processing procedure. In this processing, skin color is extracted by threshold using HSV color space, and fine noise is removed by expansion / shrinks processing. Next, in order to remove a relatively large noise region, the CCL performed. If the size (number of pixels) of the connected component is less than the specified value, the component region is discarded as noise. That is, this is the HFP using foreground and background simultaneous CCL proposed in section 3.3. Finally, the remained connected component is extracted as a face candidate area (see the rectangle in FIGURE 9-4).

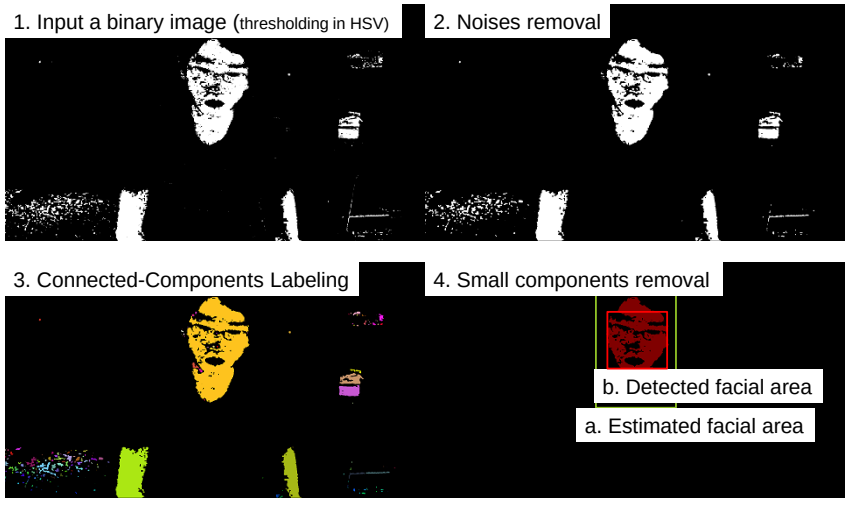

FIGURE 9: Proposed face detection procedure.

4.6.3. Evaluation

Classification by linear SVM using HOG features (conventional method) described in section 4.6.1 was implemented using an image processing C++ library called Dlib [39], and the face detection time was measured. In addition, we implemented the proposed method described in section 4.6.2 on a CPU and a GPU. On the CPU, the raster type was adopted for the CCL because the raster type was faster than the propagation type as described in section 4.1.

Image size	Processing time (ms)			Speedup ratio		
	Conv.	Proposed		(1) / (2)	(2) / (3)	(1) / (3)
	CPU		GPU			
	(1)	(2)	(3)			
864 x 480	169.4	11.8	4.9	14.4	2.4	34.6
1,280 x 720	385.4	24.8	7.7	15.5	3.2	50.1
1,920 x 1,080	834.4	56.4	14.0	14.8	4.0	59.6

TABLE 7: Processing time and speedup rate.

TABLE 7 shows the face detection time and the speed-up rate when the input image size is changed. It was confirmed that the face detection time increased in proportion to the input image size. The face detection processing by HOG + SVM took about 1 second, which is far from real-time processing of about 30 fps. With the proposed method, the face detection process using the FHD image as input is completed in about 60 ms even on the CPU, which is about 14 times faster than the conventional method.

Here, the huge processing time, which is the drawback of face detection using HOG features and SVM, was decreased by combining relatively simple image processing techniques. Furthermore on GPU, by using a parallel label rewriting algorithm of CCL for GPGPU, processing was completed in about 14 ms per FHD image, and about 71.4 fps was achieved. This is about 60 times faster than the conventional facial detection method.

Regarding the facial detection accuracy, the conventional method using SVM can detect 100% faces for 30,000 images, whereas the proposed method can detect 27,761 images, which is 92.5% accurate. Although the proposed method has a detection accuracy of 90% or more, in order to make this method more practical, it is considered that it is better to use this method to reduce face candidate regions.

5. CONCLUSIONS

In this paper, we proposed 1) a rewrite algorithm for continuous label ID for GPGPU, and 2) a parallel algorithm for hole filling using simultaneous connected components labeling for foreground and background, and its optimization method.

In 1), an algorithm that rewrites for SIMD operation the label ID into a serial number achieved a speed increase of approximately 9.2 times on average compared to the conventional method. As a result, it is now possible to process by only all label search, without performing all pixel search described in section 3.4. In 2), two CCLs were reduced to one by extracting the foreground and background at the same time, and as a result, the idle time of the core in SIMD operations such as GPGPU was reduced. Furthermore, we proposed a hole filling process using foreground and background simultaneous CCL, and a computational reduction method using the properties of label ID.

These proposed methods ware implemented on TX2 and when a practical leaf image is used as the input image, the processing time is about 1.13-1.27 times faster than the conventional GPU execution. In addition, it is about 1.40-2.46 times faster than the conventional CPU execution. The processing time of the proposed method for full HD leaf images is about 18.6 milliseconds, and the throughput is about 53.8 fps. By applying this proposed method to object detection, we are able to achieve 60 times faster than the CPU implementation of the conventional method.

In this evaluation experiment, two artificial images and five types of leaf images were used, but in order to strengthen the results of this research, evaluation experiments with more diverse images are necessary. In the experiments so far, the number of scans for propagating the minimum label ID was fixed, but when many types of images are targeted, it is necessary to dynamically determine the number of scans. That is, once every several times of scanning, it is necessary to judge whether the propagation is completed and terminate the scanning. Furthermore, it is necessary to evaluate the trade-off between the judgement load and the reduction in the number of scans.

The connected component labeling process for GPGPU has little room for optimization, and it has been difficult to achieve higher speeds. However, we focused on wasted idols in the hole filling process, proposed a method to reduce and supplement them, and realized a faster hole filling process than the conventional method.

6. REFERENCES

[1] L. He, Q. Gao, X. Zhao, et al. “The connected-component labeling problem: A review of state-of-the-art algorithms.” Pattern Recognition, vol. 70, pp. 25-43, Oct. 2017.

[2] Y. Purwar, S. L. Shah, G. Clarke, A. Almugairi, A. Muehlenbachs. “Automated and unsupervised detection of malarial parasites in microscopic images.” Malaria Journal, vol. 10, no. 364, pp. 1-10, Dec. 2011.

[3] A. Ito, Y. Aoki, S. Hashimoto. “Accurate extraction and measurement of fine cracks from concrete block surface image,”in Proc. 28th Annual Conference of the IEEE Industrial Electronics Society (IECON02), vol. 3, 2002, pp. 2202-2207.

[4] A. Rakhmadi, M.S.M. Rahim, A. Bade, et al. “Loop back connected component labeling algorithm and its implementation in detecting face.” International Journal of Computer, Electrical, Automation, Control and Information Engineering, vol. 4, no. 4, pp. 635-640, Apr. 2010.

[5] Y. Bian, F. Dong, H. Wang. “Reconstruction of rising bubble with digital image processing method,” in Proc. 2011 IEEE International Instrumentation and Measurement Technology Conference, IEEE, 2011, pp.1-5.

[6] O. St., B. Benes. “Connected component labeling in CUDA” In GPU computing gems emerald edition, W. W. Hwu, Morgan Kaufmann, 2011, pp.569-581.

[7] N. Shibata, S. Yamamoto. ”GPGPU-Assisted Subpixel Tracking Method for Fiducial Markers.” Journal of Information Processing, vol. 22, no. 1, pp. 19-28, Jan. 2014.

[8] NVIDIA. “NVIDIA JETSON The embedded platform for autonomous everything.” https:// www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/, Apr. 29, 2020 [May. 07, 2020].

[9] K. Suzuki, H. Horiba, N. Sugie. “Linear-time connected-component labeling based on sequential local operations.” Computer Vision and Image Understanding, vol. 89, no.1, pp.1-23, Jan. 2003.

[10] K. Wu, E. Otoo, K. Suzuki. “Optimizing two-pass connected-component labeling algorithms.” Pattern Analysis and Applications, vol. 12, no. 2, pp. 117-135, Jun. 2009.

[11] L. He, Y. Chao, K. Suzuki. ”A run-based two-scan labeling algorithm.” IEEE transactions on image processing, vol. 17, no. 5, pp. 749-756, Mar. 2008.

[12] S. Gupta, D. Palsetia, M. Patwary, M. Ali, A. Agrawal, A. Choudhary. “A new parallel algorithm for two-pass connected component labeling.” In Proc. 2014 IEEE International Parallel & Distributed Processing Symposium Workshops, 2014, pp. 1355-1362.

[13] O. Kalentev, A. Rai, S. Kemnitz, R. Schneider. “Connected component labeling on a 2D grid using CUDA.” Journal of Parallel and Distributed Computing, vol. 71, no. 4, pp. 615-620, Apr. 2011.

[14] N. Shibata, S. Yamamoto, “SumiTag: Uses a less noticeable AR marker and GPGPU Read method.” IPSJ DPS research report, vol. 2011-DPS-149, no.7, pp.v1-9, in Japanese, Nov. 2011.

[15] N. Shibata, S. Yamamoto, “Implementation of parallel algorithm for connected component extraction for CPU using AVX2 instruction set.” In Proc. Workshop on Multimedia Communication and Distributed Processing (DPSWS2013), 2013, pp. 300-307, in Japanese.

[16] H. Kawada, T. Kamiya, Y. Marutani, “A Method of Strain Inspection of Automotive Rearview Mirror by Image Processing.” IEICE Transactions D, vol. J83-D2, no. 3, pp. 947-956, in Japanese, Mar. 2000.

[17] T. Oshige, “Automatic inspection technology for quality evaluation of steel products.” Measurement and control, vol. 55, no. 3, pp. 228-233, in Japanese, Mar. 2016.

[18] H. Ogawam D. Sakai. “Diagnosis of cucumber leaf disease by image processing.” Bulletin of Aichi Univ. of Education, 58 (Natural Sciences), pp. 13-19, in Japanese, Mar. 2009.

[19] A. Hashidume, R. Suzuki, H. Yokouchi, H. Horiuchi, S. Yamamoto, “Red blood cell automatic discrimination algorithm and its evaluation.” Medical electronics and biotechnology, vol. 28, no. 1, pp. 25-32, in Japanese, Jan. 1990.

[20] J. Hasegawa, K. Mori, J. Toriwaki, Y. Yasuno, K. Katada, “Extraction of Lung Cancer Candidate Regions from Chest Sequential CT Images by 3D Digital Image Processing.” IEICE Transactions D, vol. J76-D2, no. 8, pp.1587-1594, in Japanese, Aug. 1993.

[21] J. Masumoto, M. Hori, Y. Sato, T. Murakami, T. Kamikou, H. Nakamura, S. Tamura, “Study on automatic liver tumor extraction from X-ray CT images.” IEICE Transactions D, vol. J83-D2, no. 1, pp. 219-227, in Japanese, Jan. 2000.

[22] J. Masumoto, M. Hori, Y. Sato, T. Murakami, T. Kamikou, H. Nakamura, S. Tamura, “Automatic liver region extraction from multi-slice CT images.” IEICE Transactions D, vol. J84-D2, no. 9, pp. 2150-2161, in Japanese, Sep. 2001.

[23] Y. Hayase, Y. Mekata, K. Mori, J. Hasegawa, J. Torwaki, M. Mori, H. Natori, “A method for detecting multiple nodules from 3D chest X-ray CT images.” IEICE Transactions D, vol. J87-D2, no. 1, pp. 219-227, in Japanese, Jan. 2004.

[24] Y. Hirano, J. Hasegawa, J. Toriwaki, H. Daimatsu, K. Eguchi, “Interactive lung tumor region extraction from 3D chest CT images and its application to malignant differentiation.” IEICE Transactions D, vol. J87-D2, no. 1, pp. 237-247, in Japanese, Jan. 2004.

[25] H. Furukawa, K. Ueda, R. Tachibana, N. Kido, “Extraction of liver region from 3D abdominal CR image using CT value distribution information and template image.” Computer Assisted Diagnostic Imaging Society, vol. 9, no. 3, pp. 27-35, in Japanese, Nov. 2005.

[26] A. Seal, S. Gangulyb, D. Bhattacharjee, M. Nasipuri, D.K. Basu. “Minutiae based thermal human face recognition using label connected component algorithm.” Procedia Technology, vol. 4, pp. 604-611, Feb. 2012.

[27] A. Hemlata, M. Motwani. “Face detection by finding the facial features and the angle of inclination of tilted face.” International Journal of Computer Science Issues (IJCSI), vol. 10, issue 2, pp.472-479, Mar. 2013.

[28] H. Arunachalam, M. Motwani. “Image segmentation for the extraction of face using haar like feature”, Int. Arab J. Inf. Technol, vol. 13, no. 6A, pp.951-958, Dec. 2016.

[29] H. Kizuna, H. Sato. “Accelerating Facial Detection for Improvement of Person Identification Accuracy in Entering and Exiting Management System.” In Proc. Sixth International Symposium on Computing and Networking Workshops (CANDAR), 2018, pp. 202-208.

[30] H. Kizuna, H. Sato. “Acceleration of Face Detection for Improving Person Identification Accuracy in Entrance / Exit Control System.” IEICE research report of Image Engineering, vol. 117, no. 484, pp. 229-234, in Japanese, Mar. 2018.

[31] T. Miyake, S. Haruta, S. Horihata, “Gaze determination method using features that do not depend on face orientation.” IEICE Transactions D, vol. J86-D2, no. 12, pp. 1737-1744, in Japanese, Dec. 2003.

[32] D. D. Sidibe, P. Montesinos, S. Janaqi. “A simple and efficient eye detection method in color images.” in Proc. International Conference Image and Vision Computing New Zealand 2006, pp. 385-390.

[33] Y. Saito, Y. Kenmochi, K. Kotani, “Extraction and removal of eyeglass frame area in face image using parametric eyeglass frame model.” IEICE Transactions D, vol. J82-D2, no. 5, pp. 880-890, in Japanese, May. 1999.

[34] T. Fuda, S. Omachi, H. Aso, “Recognition of Line Graph Images in Documents by Tracing Connected Components.” IEICE Transactions D, vol. J86-D2, no. 6, pp. 825-835, in Japanese, Jun. 2003.

[35] O. Shiku, A. Nakamura, “Character line extraction from scene image using LoG filter.” IEICE Transactions D, vol. J87-D2, no.8, pp. 1735-1739, in Japanese, Aug. 2004.

[36] Y. Aramaki, Y. Matsui, T. Yamasaki, K. Aizawa, “Character region detection in comics based on connected components and region classification.” IEICE Transactions A, vol. J100-A, no.1, pp. 3-11, in Japanese, Jan. 2017.

[37] N. Dalal, B. Triggs, “Histograms of oriented gradients for human detection.” in Proc. International Conference on Computer Vision and Pattern Recognition (CVPR), 2005, vol. 1, pp. 886-893, 2005.

[38] V.N.Vapnik. “Statistical Learning Theory”, Wikey, New York, 1988.

[39] D. E. King. “Dlib C++ Library.” http://dlib.net/, Dec.14.2019 [Mar.11.2020].