IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 2, 2018, accepted September 12, 2018, date of publication September 28, 2018, date of current version October 25, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2872452

Efficient Parallel Connected Component Labeling With a Coarse-to-Fine Strategy

JUN CHEN, KEISUKE NONAKA, HIROSHI SANKOH, RYOSUKE WATANABE, HOUARI SABIRIN, AND SEI NAITO

Ultra-realistic Communication Group, KDDI Research, Inc., Fujimino 3568502, Japan

Corresponding author: Jun Chen (ju-chen@kddi-research.jp)

ABSTRACT This paper proposes a new parallel approach to solve connected components on a 2-D binary image. The following strategies are employed to accelerate neighborhood exploration after dividing an input image into independent blocks: 1) in the local labeling stage, a coarse-labeling algorithm, including row-column connection and unification, is applied first to reduce the complexity of an initialized local label map; a refinement algorithm is then introduced to merge separated sub-regions from a single component; and 2) in the block merge stage, we scan the pixels on the block boundary instead of solving the connectivity of all the pixels. With the proposed method, the length of label-equivalence lists in both the local labeling stage and global labeling stage are compressed and the number of memory accesses is reduced. Thus, the efficiency of connected component labeling is improved. The proposed strategies are illustrated using 4-neighbor connectivity, and the case of 8-neighbor connectivity is also discussed. The YACCLAB data sets, including both synthetic and real images, are used to evaluate the new algorithm and compare it to existing algorithms. The comparative results show that the proposed new algorithm outperforms the other approaches in both the 4-neighbor connectivity and 8-neighbor connectivity cases.

INDEX TERMS Connected component labeling, parallel computation, real-time image processing, optimization method.

I. INTRODUCTION

Connected component labeling (CCL) is a task that gives a unique ID to each connected region in an image. It means that the input data are clustered as separate groups where the elements from a single group share the same ID. As a basic data clustering method, CCL is widely used as a tool for object detection and classification in the field of computer vision and image processing [1]–[5]. Song *et al.* [6] presented a motion-based skin region of interest detection method using a real-time CCL algorithm to reduce execution time. A fast 3D shape measurement technique using blink-dot projection patterns has been reported [7], [8] that utilizes a CCL algorithm to compute the size and location of each dot on captured images. P. Guler et al. proposed a real-time multi-camera video analytics system [9] employing CCL to perform noise reduction. Acting as a fundamental operation in applications, especially in real-time applications, the acceleration of CCL is an important task [10], [11].

Numerous studies have proposed ways to speed up CCL. The proposed solutions implemented on CPU can be summarized into two classes: label propagation and label equivalence algorithms [10]. The approaches [12], [13] based on label propagation often find an unlabeled pixel using raster scan and give it an unused label; the label is then propagated to all the pixels in the same region in an irregular approach such as tracing the object's contour [14]. These approaches are not suitable for parallel and hardware implementation because of the existence of the irregular scan. The methods [15]–[18] based on label-equivalence solve the CCL issue with multiple raster scans. A provisional label, often associated with the pixel position in the image, is assigned to each pixel in the first scan; the label-equivalence lists are constructed based on pixel connectivity and resolved with root-find algorithms in the subsequent steps. Since the pixels are processed in a regular way, it is feasible for these methods to be extended to parallel and hardware implementation [19], [20].

Until recently, the use of GPUs with interfaces such as CUDA [21] or OpenCL [22] have found countless applications in both industry and academia. The parallel extension and improvement of serial CCL algorithms are significant advances toward enhancing real-time performance.

For algorithms implemented on GPUs, data parallelization across multiple processors [23], [24] plays an important role in computing with multiple processing elements. Generally, the different data parallelization approaches lead to different computation algorithms. Consequently, the reported solutions for CCL on GPUs can be classified into three types: pixel-based algorithms, block-based algorithms, and line-based algorithms. The first type extends label-equivalence-based algorithms into parallel ones directly by considering each individual pixel or the pixels in a small group as a computation unit. The other two types first divide the input image into independent sections, blocks or lines, and then perform local labeling and section merge to solve the CCL issue.

In this study, we propose a block-based solution to reduce the number of iterative operations by exploring the benefit of two-dimensional pixel distribution. Its main contributions are: (1), in the local labeling stage, row-column unification is performed using shared memory to reduce the complexity of an initialized local label map. It shortens the path to reach the root of a pixel and thereby reduces the number of atomic operations. (2), in the block merge stage, connectivity analysis is conducted for the pixels on the block boundary, but not for all the pixels, to reduce the number of memory accesses. In the following sections, we will outline our method in the case of 4-neighbor connectivity, prove the positive effects of the coarse-to-fine strategy, and demonstrate its performance by comparing it to existing algorithms across a range of test datasets.

II. PREVIOUS WORK

A. PIXEL-BASED CCL ALGORITHM

Label-equivalence [25] is an algorithm that records the lowest label that each label is equal to form label-equivalence lists in the first pass, and resolves the equivalence in the other scans. In many cases, the first pass of this method generates several disjoint equivalence lists for a single connected region so that it has to scan the input iteratively to find a final label map. Its operation model is expressed in Fig. 1. Jung and Jeong [26] solved the CCL issue by interactively executing six phases, comprising initialization, scan, analysis, link, label, and rescan. In the scan and link phases, they introduce specific masks to construct label-equivalence lists. In the analysis and label phases, they find the roots by tracing each list. Kalentov *et al.* [27] improved the label-equivalence technique in terms of memory consumption and processing steps, which removed the reference array and atomic operations in the scan phase. Soh *et al.* [28] proposed a direction-based searching method that obtains the minimum label by tracing the branches derived from a focused pixel in four or eight directions. Block-equivalence [29] is another extension of the label-equivalence solution. It uses a superpixel block instead of individual pixels taking into consideration that the pixels located in a 2×2 block share the same label. It is an effective approach because it reduces the number of candidate pixels for the connectivity test.

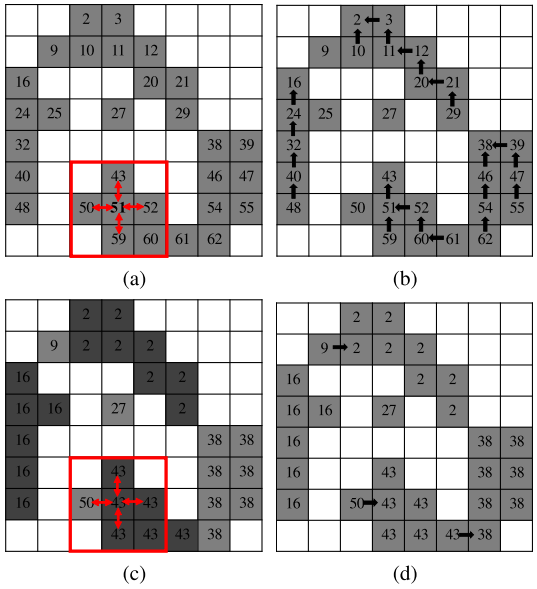

FIGURE 1. Operation model of [25]. (a) The label map is initialized with the raster scan order. Each pixel examines its neighbors to obtain the lowest label. (b) The lowest label is put into a list to form a label-equivalent chain. (c) The label map is updated by solving the label equivalence. It can be seen that a single connected region is divided into several disjoint equivalence lists. Each pixel examines its neighbors again after update. (d) The lowest label is put into a list to form a new label-equivalent chain.

The main drawback of these pixel-based algorithms is that a single label-equivalence list cannot be constructed for one connected component in one scan. Consequently, the kernels of these algorithms should be spawned several times to guarantee that no disjoint equivalence lists exist for a single region. Even though some of them reduce the number of iterations at some level, they still need to scan the input image multiple times. Moreover, the iterations might vary dramatically in different images.

B. BLOCK-BASED CCL ALGORITHM

The parallel version of the union-find algorithm [31] is presented by Oliveria and Lotufo [32]. They executed two merges successively, local and global merges, to overcome the drawback that a long path may need to be followed to reach the root of a label equivalence list. Although this algorithm outperforms most of the pixel-based CCL algorithms because all the kernels are spawned only once, searching for the root of a specific pixel is computationally heavy. Stava and Benes [30] designed a solution in a similar manner. In the local merge stage, they improved the label-equivalence algorithm by implementing all iterations inside the kernel, such that no synchronization between host and device is required. In the global merge stage, they use the connectivity of the border elements of two neighboring blocks to merge their equivalence lists. It is necessary to perform global merge several times to guarantee that all equivalence lists are merged. Its operation model is expressed in Fig. 2. Kumar *et al.* [33] implemented the CCL algorithm using a divide and conquer technique [34] on CUDA, which solves

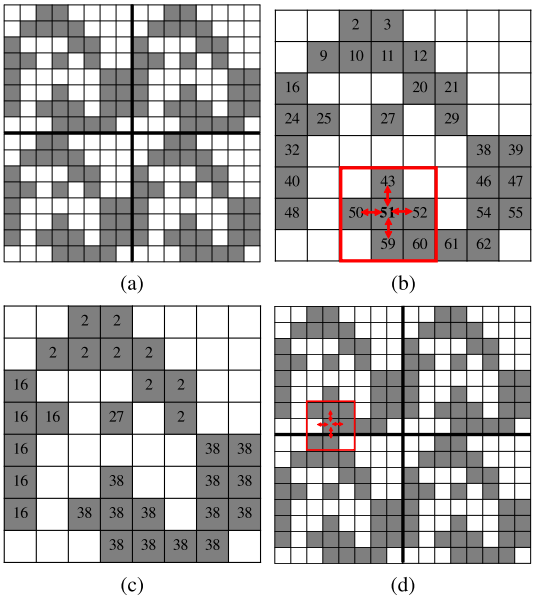

FIGURE 2. Operation model of [30]. (a) An input image is divided into independent blocks. Each block is assigned to different GPU processors to utilize shared memory to accelerate the processes of block unification. (b) Each block is initialized individually and unified by the label-equivalence method. (c) A local label map after block unification. (d) The label-equivalence method is applied to boundary elements iteratively to merge the blocks.

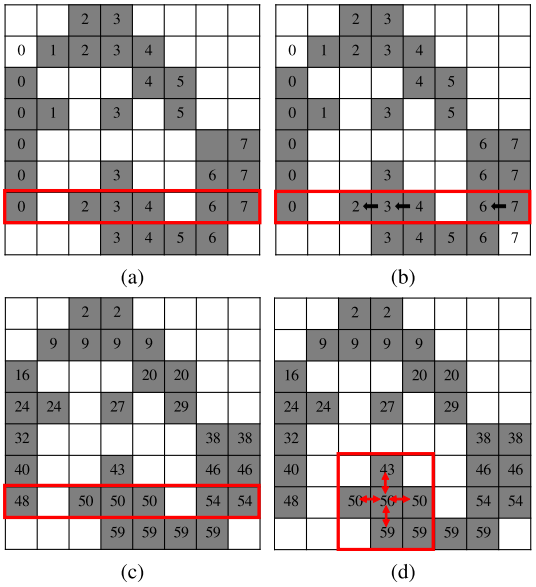

FIGURE 3. Operation model of [36]. (a) An input image is divided into independent lines, each line is assigned to different GPU processors to utilize shared memory to accelerate the processes of line unification. (b) Each pixel examines its neighbors on the left to form horizontal label-equivalence lists. (c) Each line is unified by a root-find algorithm, and local labels are converted to global labels. (d) The label-equivalence method is applied to the entire input image to merge each line.

the local connection using the Floyd-Warshall algorithm [35] and merges blocks by considering three different cases. Here, the various processing approaches for the three cases lead to thread divergence thus limiting the performance.

C. LINE-BASED CCL ALGORITHM

Chen *et al.* [37] proposed a two-scan approach, an extension of the stripe-based CCL method [38], to carry out stripe extraction and stripe union, respectively. The first scan can run in parallel by using shared memory, while the second scan is a sequential operation. ACCL [39] is another parallelization algorithm that decomposes the image into rows. By defining a span as a group of pixels that are located contiguously in a row with the same intensity, it spawns two kernels, find and merge spans, to label an input image. The involvement of dynamic parallelism means that good performance can be achieved. However, it is not suitable for processing large images because there is a limitation on the number of threads in one block [40]. Yonehara and Aizawa [36] proposed a line-based solution that accelerates the local labeling phase by conducting row unification using shared memory. The absence of the union-find algorithm makes it label each individual line efficiently in the first scan, while the global merge follows the method of the label-equivalence approach. Its operation model is expressed in Fig. 3.

III. ALGORITHM AND IMPLEMENTATION

We assume that a pixel in an image has three attributes, comprising position, intensity, and label. Position means its raster scan order in row-major form, which can be expressed by $P(x, y) = x + y * W$. Here (x, y) is its 2D coordinate in the image. (H, W) is the resolution of the image. Intensity is the color intensity of a pixel, which can be expressed by $I(x, y)$. In our implementation, $I(x, y) = 1$ when a pixel belongs to the foreground, and $I(x, y) = 0$ when it is background. Label $L(x, y)$ is what we should find to depict each connected region.

There are three steps in our method to solve the component labeling of an image. In the first step, we divide the input image into blocks and perform local labeling with a coarse-to-fine strategy. In the second step, we examine the connectivity of pixels on the block boundary to form global label-equivalence lists. In the last step, the final label map is obtained by solving equivalence using a root-find algorithm.

A. LOCAL LABELING WITH COARSE-TO-FINE STRATEGY

The first step, local labeling with a coarse-to-fine strategy, consists of four phases, comprising initialization, coarse labeling, refinement, and ID conversion.

1) INITIALIZATION

In our algorithm, each pixel should first be assigned a provisional label so that a connection-list can be constructed and solved. We define $L(x, y) = P(x, y)$ so that the provisional label of (x, y) corresponds to its 2D position. In this way, the root of a label equivalence list is the element whose label equals the raster scan order of the element itself. The process of initialization is illustrated by Fig. 2 (a) and (b) where (a) shows that a $(H, W) = (16, 16)$ binary image is split

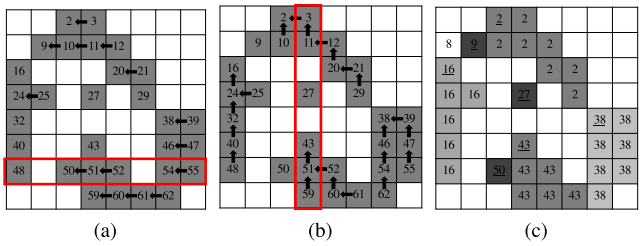

FIGURE 4. Coarse labeling. (a) Label-equivalence list after row scan. (b) Label-equivalence list after column scan. (c) Local label map after row-column unification.

into four sub-images with a resolution of $(H_s, W_s) = (8, 8)$; (b) presents an example of an initialized local label map. The grey elements on the image, $I(x, y) = 1$, express foreground pixels. In our CUDA implementation, we dispatch the sub-images to various GPU thread blocks where the threads can cooperate with each other using shared memory and can be synchronized [41]. The provisional labels and pixel positions are associated with the thread ID within a thread block. Thread synchronization is necessary because the buffer could not be initialized by multiple threads at the same time.

2) COARSE LABELING

In an initialized local label map, the provisional label of the left pixel and that of the upper pixel are always smaller than the label of a target pixel, while the upper one is always the minimum. Based on this fact, we scan rows and columns successively to build a coarse label-equivalence list. In the case of a row scan, we associate two horizontal neighbor pixels by updating the label of the right one with the label of its left neighbor $L(x, y) = L(x - 1, y)$ if both of them are foreground $I(x, y) = I(x - 1, y) = 1$. Label-equivalence trees are constructed for consecutive foreground pixels in each row as presented in Fig. 4 (a). The scanning approach in the vertical direction is performed in the same manner where the association in the horizontal direction is replaced by the association in the vertical direction if all three pixels are foreground $I(x, y) = I(x - 1, y) = I(x, y - 1) = 1$. Fig. 4 (b) presents a demonstration of coarse label-equivalence lists after row-column scan. The same result can be achieved by comparing the labels of the above-mentioned three pixels directly at the same time. However, we find that the proposed method is faster because it does not involve branch divergence and boundary-related operations. Furthermore, this method records the lowest neighbor label that the label is equivalent to but does not attempt to record the entire equivalence. Its memory access complexity is reduced due to the utilization of shared memory. Fig. 4 (c) illustrates the roots of each list and the coarse local label map after solving equivalence. It should be noted that this step can not provide a complete segmentation but splits a connected region into several groups.

3) REFINEMENT

This phase is a task that involves to merging the segments that belong to a single region. As shown in Fig. 5 (a), three isolated

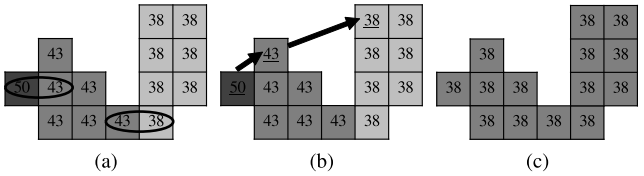

FIGURE 5. Refinement. (a) A single region segmented into three disjoint components. (b) A label-equivalence list was constructed by another row-scan. (c) Unified label map of a single region.

sub-regions exist in one connected region. The pixels in the ellipse are the branch dividing points that lead connected pixels to different label-equivalence lists. The operations in coarse labeling define that the label of $p(x, y)$ associates with the label of $p(x, y - 1)$ if pixels $p(x, y)$, $p(x - 1, y)$, and $p(x, y - 1)$ are foreground. This associativity separates three connected pixels into two groups $G(1) = \{p(x - 1, y)\}$ and $G(2) = \{p(x, y), p(x - 1, y)\}$. It is found that the branch dividing points are in the horizontal direction so that the sub-regions can be merged by performing another row unification. New label-equivalence lists are constructed for consecutive foreground pixels in each row. When two horizontal neighbor pixels have different labels, we put the label into the list if it is lower than the label currently in the list. Fig.5 (b) shows the label-equivalence list of the example. It demonstrates that three isolated sub-regions are associated with each other by a list with three elements, which is a major reduction compared to the other methods. Finally, the region is unified using a root-find algorithm as shown in Fig. 5 (c). It should be noted that atomic operations are necessary here because the same equivalence list may be updated by multiple threads at the same time.

4) ID CONVERSION

The final local labeling phase is an ID conversion process that converts the local label to a global label and transfers the result to global memory. The global label identifies the raster scan order of a pixel in the entire image as shown in Fig. 6.

B. BLOCK MERGE WITH BOUNDARY ANALYSIS

In the block merge phase, we perform a connectivity test for the pixels on the block boundary to merge a single connected component that exists in different blocks. Assuming the block configuration of local labeling is $\{b_x, b_y, 1\}$, the number of border pixels along the $x-$axis N_x and the number of border pixels along the $y-$axis N_y can be determined as follows:

$$N_x = \lfloor H \,/\, b_y \rfloor * W - W, \tag{1}$$

$$N_y = \lfloor W \,/\, b_x \rfloor * H - H, \tag{2}$$

Here, $\lfloor x \rfloor$ means the largest integer smaller or equal to x. It is found that the number of candidate pixels for the connectivity test get reduced by $\frac{H \times W}{N_x}$ times for boundary analysis along the $x-$axis and $\frac{H \times W}{N_y}$ times for boundary analysis along the $y-$axis.

Similar to coarse labeling, we scan the vertical boundary and the horizontal boundary successively. If two neighbor

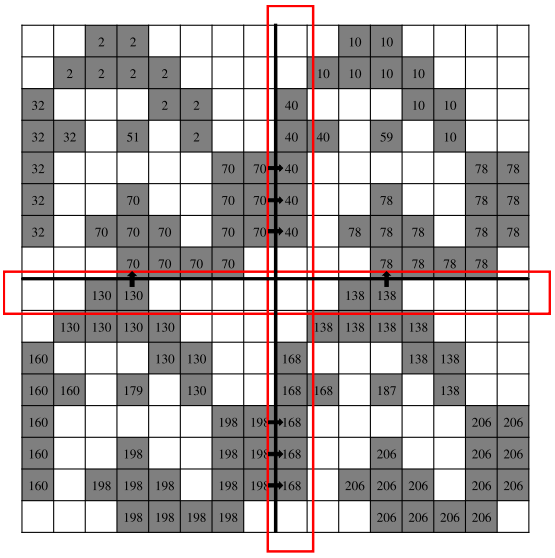

FIGURE 6. Label map after local labeling.

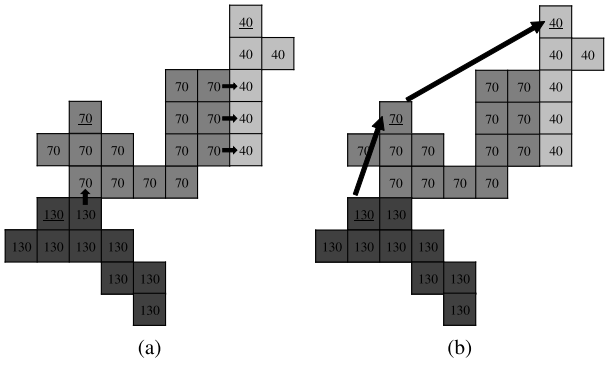

FIGURE 7. Global label-equivalence construction. (a) A single region with three disjoint components. (b) A global label-equivalence list created by connecting the roots of three components.

pixels are foreground, we put the smaller label into the label equivalence list. As shown in Fig. 6, the pixels in rectangles are boundary pixels, and the arrows show the association between pixels. Fig. 7 (a) presents a connected region that is composed of three sub-regions from three blocks. It can be seen that the boundary analysis provides the opportunity to connect the three sub-regions together. Fig. 7 (b) gives a global label-equivalence list by putting the root of each sub-region into the list. Here, the depth of the list is three, showing that the depth of the global label-equivalence list is not related to the image resolution but is related to the number of blocks. In our implementation, $\max\{N_x, N_y\}$ threads should be invoked to integrate the boundary analysis along the $x-$ and $y-$axes into one kernel.

C. UPDATE GLOBAL LABEL MAP

The final global label map represents the complete segmentation of an input image where every equivalence list corresponds to an unbroken connected component. The independent blocks are associated as an entirety after boundary analysis, such that the roots of global label-equivalence lists can be obtained by using a root-find algorithm.

IV. COMPARATIVE EVALUATION

In order to demonstrate the performance of our method, we compare it to the following approaches.

- C2FL as our proposed method.
- RC2FL as a revised version of our method with coarse labeling only along row.
- CC2FL as a revised version of our method with coarse labeling only along column.
- NC2FL as a revised version of our method with no coarse-to-fine strategy. It is a fact that the local merge process of NC2FL is the same as that of UF [32].
- LE [27] as a conventional pixel-based label equivalence solution.
- SMCCL [30] as a more recent block-based method using shared memory.
- UF [32] as a more recent block-based method using a union-find method.
- LUF [36] as a more recent and fast representative of the line-based method.

There are two kinds of comparative experiments. The first is an evaluation of the effectiveness of the coarse-to-fine strategy which compares C2FL to RC2FL, CC2FL, and NC2FL. It should be noted that local labeling works correctly if refinements along both rows and columns are applied even without coarse labeling. When each individual pixel is considered as a sub-region of a single connected region, the third phase, refinement, is able to generate an entire local label-equivalence list. In fact, this is what UF [32] used in the local labeling stage. The significance of coarse labeling is that it allows local merge to be performed efficiently. The second is a comparison with other existing CUDA-based algorithms. The execution times of C2FL, LE, SMCCL, UF, and LUF for datasets [42] are listed.

All the experiments were performed on a PC Intel(R) Core(TM) i7-6700K CPU, 4.00 GHz & 4.00 GHz, 32.0 GB RAM, NVIDIA Geforce GTX 1070 with Windows 7 Professional Service Pack 1. All the algorithms were implemented in C++ language using OpenCV 2.4.13 and CUDA 8.0. For the reported execution time, the average over 100 runs on every image is collected to remove any fluctuations caused by the other tasks executed by the operation system. Meanwhile, we touched all the memory before use to avoid counting the allocation time. All the algorithms were implemented based on 4-neighbor connectivity.

A. EFFECTIVENESS OF COARSE-TO-FINE STRATEGY

There are two significant factors in one thread block that affect the efficiency of local labeling, the number of iterations and atomic operations. Iteration refers to the process of iterating an operation such as tracing a label-equivalence list to find its root. Most of the algorithms that are expressed in C++ language take only a few lines. However, there may be thousands of instructions that are executed on hardware.

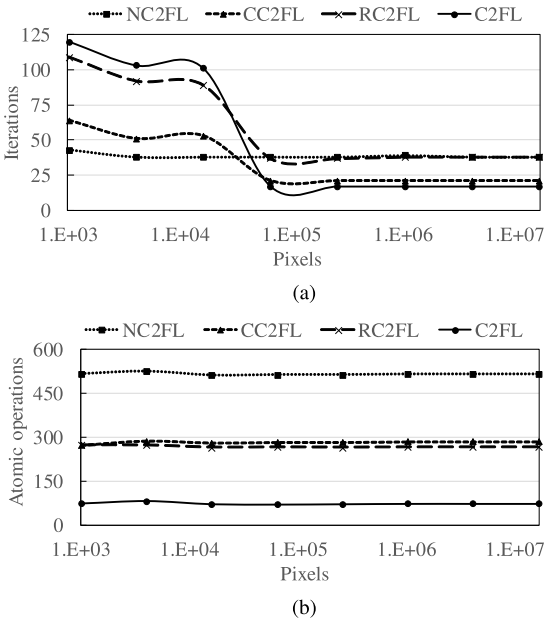

FIGURE 8. Iteration and atomic operation versus image size. (a) Iteration versus the image size. (b) Atomic operation versus the image size.

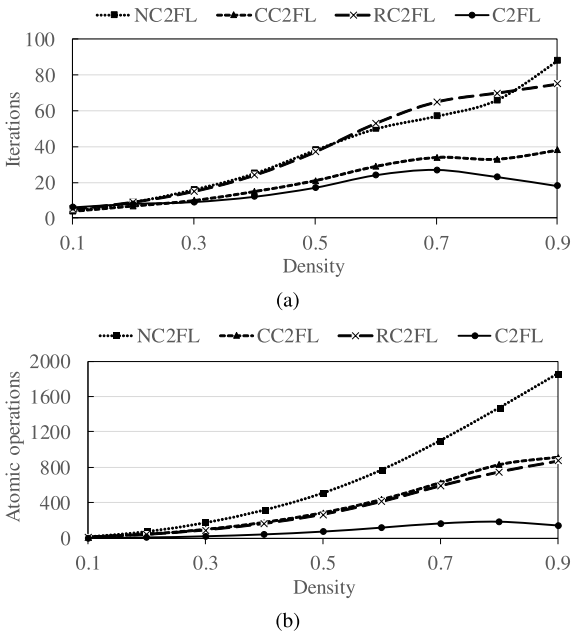

FIGURE 9. Iteration and atomic operation versus foreground density. (a) Iteration versus foreground density. (b) Atomic operation versus foreground density.

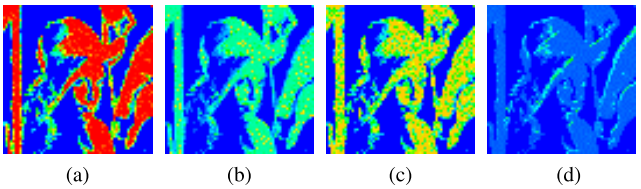

FIGURE 10. Number of iterations expressed by color map. (a) NC2FL. (b) CC2FL. (c) RC2FL. (d) C2FL.

Generally, the number of iterations reflects the number of instructions and the program execution time. Atomic operations are a kind of processing performed without interference from any other threads. They are often essential for multi-threaded applications to prevent race conditions, especially when different threads attempt to modify the same memory address. If two or more threads perform an atomic operation at the same memory address at the same time, those operations will be serialized. This means that the more atomic operations there are the slower the execution.

In our evaluation, the CUDA thread block was configured as $\{b_x = 16, b_y = 16, 1\}$. We use a set of square binary images with various resolutions and random noise to show the difference in the number of iterations as well as the number of atomic operations among C2FL, RC2FL, CC2FL, and NC2FL. There are nine different foreground densities, from 0.1 to 0.9, and eight resolutions, from a low resolution of 32×32 pixels to a maximum resolution of 4096×4096 pixels. There are ten images for every couple of size and density. The experiments provide us with an opportunity to evaluate the performance of the coarse-to-fine strategy in terms of the scalability of the number of pixels and the scalability of the density of connected regions.

Fig. 8 shows how the iterations and atomic operations of different algorithms change with images of increasing size. Here, the foreground density is 0.5, which remains the same in all the images. The reported results were the average of all the launched thread blocks of 100 runs of each algorithm. As presented in (a) of Fig. 8, it can be seen that the iterations for coarse-labeling, regardless of whether it is full coarse-labeling or partial coarse-labeling, are computationally heavier than those for NC2FL when the number of pixels is less than 65535 (256×256 image), while these iterations are equal to or less than those of NC2FL when the number of pixels of an image exceeds 65535. The phenomenon is explicable because the local label-equivalence list of a low-resolution image is short. Under these circumstances, compression by coarse labeling can not reduce but rather increases the number of iterations. The linear independence of the number of atomic operations with respect to image size can be observed in (b) of Fig. 8. The proposed algorithm always takes the fewest atomic operations to segment a local image.

Fig. 9 highlights the behavior of the algorithms when the foreground densities of a 2048×2048 image are varied. It can be proved that both of the factors have a significant linear correlation with the foreground densities, and our method has the best performance among all the densities. This result is logical because coarse labeling reduces computation complexity by reducing the complexity of an initialized local label map.

The third experiment demonstrates the efficiency of coarse labeling using a binary Lena image with a size of 2048×2048. Fig. 10 shows the color map of the maximum iterations of each thread block of each algorithm where the darker color

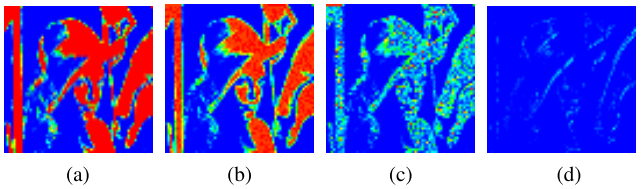

FIGURE 11. Number of atomic operations expressed by color map. (a) NC2FL. (b) CC2FL. (c) RC2FL. (d) C2FL.

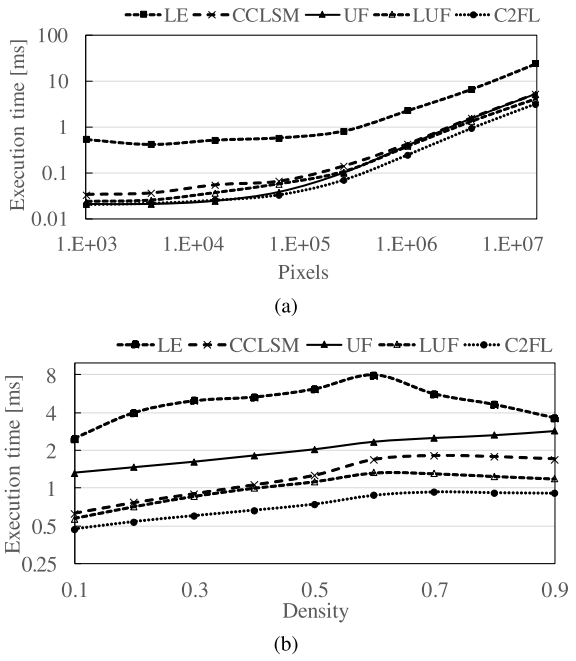

FIGURE 12. Execution time versus foreground density and image pixels. (a) Execution time versus image pixels. (b) Execution time versus foreground density.

represents more numerous iterations. It indicates that our proposed method solves the CCL issue with the fewest iterations. Fig. 11 expresses the number of atomic operations in the same manner. With regard to NC2FL, CC2FL, and RC2FL, we find that most of the race conditions occur on the blocks holding pixels from a flat foreground region. Nevertheless, there is no risk of a race condition for the blocks in C2FL.

B. COMPARISONS WITH EXISTING ALGORITHMS

We first evaluate the algorithms using the same synthetic images. Fig. 12 (a) shows how the algorithms work with images of increasing size. The execution time of all algorithms increases linearly with the expansion of input images. Our method is proved to be scalable and able to outperform all the other methods over all the sizes. Another experimental result, shown in Fig. 12 (b), highlights the efficiency of the algorithms with images of various foreground densities. It indicates that the computation is efficient when the foreground density is low or high while the worst case appears around the middle densities. Lower or higher densities present simple connections and consequently

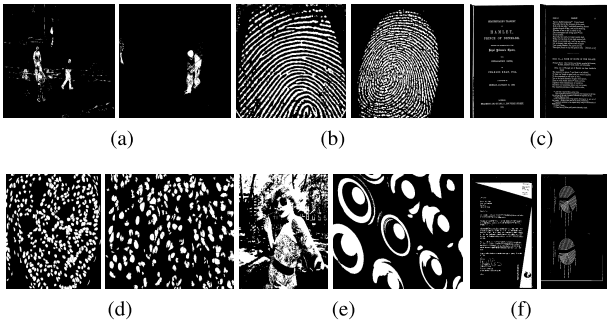

FIGURE 13. Sample images from the YACCLAB datasets. (a) 3DPeS. (b) Fingerprints. (c) Hamlet. (d) Medical. (e) MIRflickr. (f) Tobacco800.

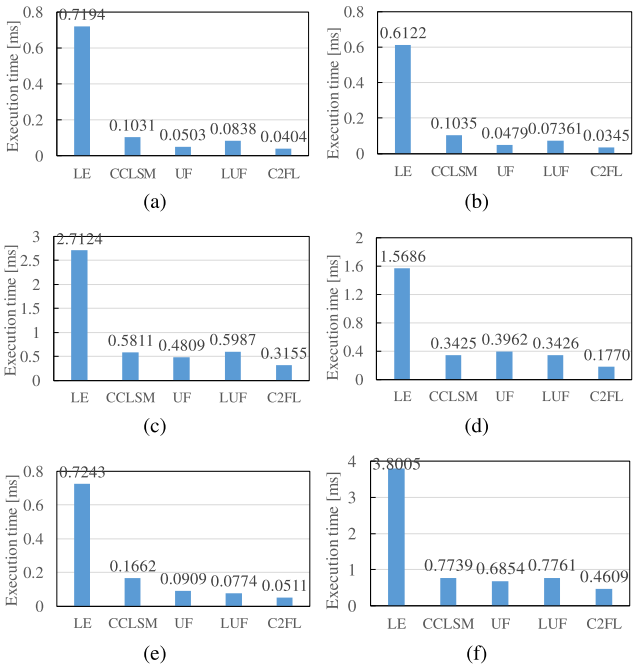

FIGURE 14. Average run-time tests on the datasets with 4-neigbor connectivity. (a) 3DPeS. (b) Fingerprints. (c) Hamlet. (d) Medical. (e) MIRflickr. (f) Tobacco800.

less computation, while the middle densities present complex connections. It can be inferred that our approach has the best performance among all the densities. It is able to label a 2048×2048 image with arbitrary density within 1.2 ms

We also run each algorithm on the YACCLAB datasets [42] to prove the performance of our proposed method. YACCLAB is a connected component labeling benchmark that contains an open source platform and a collection of datasets. All images in YACCLAB are provided in binary PNG format with black being background and white being foreground. Fig. 13 presents sample images from the datasets. The average run-times for each dataset are shown in Fig. 14. The figure shows that the performance of LE is always the worst and that our proposed method always outperforms the others. CCLSM, UF, LUF, and C2FL outperform LE because they take advantage of shared memory as an input image is divided into small blocks. They also perform iterations inside

TABLE 1. Average execution time on YACCLAB datasets in millisecond for different block configurations.

	16×16	16×32	32×16	32×32	16×64	64×16
3DPeS	0.0404	0.0405	**0.0401**	0.0429	0.0437	0.0423
Fingerprints	**0.0345**	0.0352	0.0372	0.0374	0.0368	0.0397
Hamlet	**0.3155**	0.3330	0.3262	0.3485	0.3708	0.3506
Medical	**0.1770**	0.1854	0.1917	0.2052	0.2038	0.2132
MIRflickr	0.0511	0.0492	**0.0467**	0.0484	0.0497	0.0486
Tobacco800	**0.4609**	0.4830	0.4773	0.5071	0.5267	0.5112

the kernel and do not require synchronization between CPU and GPU. C2FL outperforms CCLSM, UF, and LUF because of the coarse labeling strategy that reduces the complexity of a local block to improve the efficiency of local merge. The experimental results also show that the proposed method can provide consistent performance over all the dataset, while the performances of CCLSM, UF, and LUF vary with the different datasets.

V. DISCUSSION

A. GPU CONFIGURATION

In a CUDA program, threads are organized in blocks. The threads in the same block can communicate with each other because they run on the same stream processor. The number of threads in a block is limited by the CUDA device architecture and available shared memory. Multiple blocks are grouped into a grid to be used for computations that require a larger number of threads. For a particular CUDA application, the configuration on the thread block is an important factor as this affects the execution time. To achieve the highest efficiency, we examine the proposed algorithm using a set of thread block configurations, specifically, thread blocks of size (16 × 16), (16 × 32), (32 × 16), (32 × 32), (16 × 64), and (64 × 16). The average execution time of these cases on the YACCLAB datasets are provided in Tab. 1. Comparing these results, the first configuration (16 × 16) attains the highest performance on the Fingerprints, Hamlet, Medical, and Tobacco800 datasets. The fastest performance on the 3dpes and Mirflickr datasets is the third configuration (32 × 16). These results indicate that the performance for a specific configuration varies depending on the foreground density and image resolution. Even though it is difficult to give a configuration that is able to produce the best performance for all the datasets, the first case (16 × 16) has the potential to achieve good optimization.

B. 8-NEIGHBOR CONNECTIVITY

In a 2D image, connected components are clusters of pixels with the same properties, which are connected to each other through either 4-neighbor connectivity or 8-neighbor connectivity. The pixles of 4-neighbor connectivity groups contact each other on either of their four faces, while the pixels of 8-neighbor connectivity groups are connected along a face or corner. The proposed scheme can be extended to

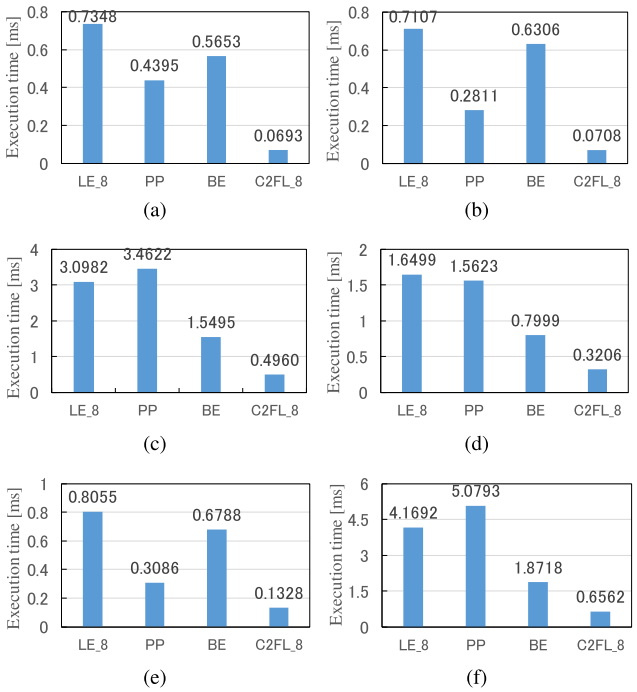

FIGURE 15. Average run-time tests on the datasets with 8-neighbor connectivity. (a) 3DPeS. (b) Fingerprints. (c) Hamlet. (d) Medical. (e) MIRflickr. (f) Tobacco800.

resolve the 8-neighbor connectivity problem with two minor modifications. The first one is that two additional scans in the diagonal direction should be processed at the local labeling stage to examine the connectivity of the corner. The second one is that, in the block merger stage, each pixel on a block boundary scans the nearest three pixels from a neighbor block to find a connection across the boundary. To evaluate the performance of our proposed method with 8-neighbor connectivity, we compared it with BE [29], LE8 [27], and a more recent CPU implementation PP [43] across the YACCLAB datasets. BE [29] is a more recent representative of the pixel-based solution using the block equivalence technique. The technique assumes the pixels in a 2 × 2 block share the same label. LE8 [27] is an extension of LE with 8-neighbor connectivity. The performance results shown in Fig. 15 indicate that the proposed method provides an approximately 5×, 7×, and 5× improvement in terms of speed over the BE, LE8, and CPU implementations, respectively. The results also

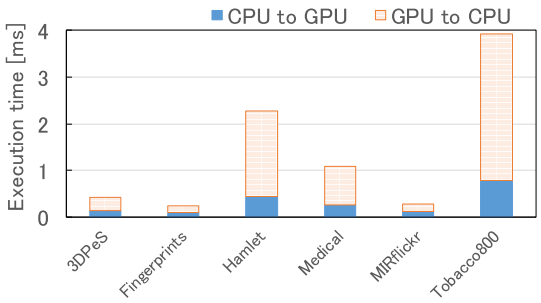

FIGURE 16. Data transfer time between GPU and CPU.

show that the 4-neighbor connectivity case is approximately 1.5-fold faster than the 8-neighbor connectivity case. The results prove that labeling an image with 8-neighbor connectivity is intrinsically more time-consuming because it requires connectivity tests in the horizontal, vertical, and diagonal directions.

C. MEMORY CONSUMPTION AND ALLOCATION

The global memory consumption of the proposed methods is the same as that of UF and LUF, which includes a space for the input image and a space for the final label map. LE and CCLSM consume more memory, which requires another buffer for the relabeling process. The shared memory consumption of the proposed methods is related to the thread block configuration. It is 1KB when $\{b_x, b_y, 1\}$ is $\{16, 16, 1\}$. The average data transfer times over 100 runs between CPU and GPU for the YACCLAB datasets are listed in Fig. 16.

VI. CONCLUSION

In this paper, we proposed a novel parallel approach with a coarse-to-fine strategy to solve the CCL issue in fewer iterations. Our method first employs coarse labeling to reduce the complexity of a local block and then applies a refinement to accelerate local labeling. In the block merge stage, we launch a low number of threads to analyze connectivity on the block boundary. The results show that the proposed method is capable of performing CCL with CUDA on GPU. We evaluated the effectiveness of the coarse-to-fine strategy and compared it with existing GPU and CPU implementations. Experimental results show that our method outperforms all existing parallel approaches. Meanwhile, it proved that our method has good scalability for various image sizes and stability for a range of various foreground densities.

REFERENCES

[1] K. Suzuki, S. G. Armato, III, F. Li, S. Sone, and K. Doi, "Massive training artificial neural network (MTANN) for reduction of false positives in computerized detection of lung nodules in low-dose computed tomography," *Med. Phys.*, vol. 30, no. 7, pp. 1602–1617, 2003.

[2] K. Suzuki, H. Yoshida, J. Näppi, S. G. Armato, III, and A. H. Dachman, "Mixture of expert 3D massive-training ANNs for reduction of multiple types of false positives in CAD for detection of polyps in CT colonography," *Med. Phys.*, vol. 35, no. 2, pp. 694–703, 2008.

[3] I. Ahmad, X. Wang, R. Li, M. Ahmed, and R. Ullah, "Line and ligature segmentation of urdu nastaleeq text," *IEEE Access*, vol. 5, pp. 10924–10940, 2017.

[4] K. Wang, C. Gou, and F.-Y. Wang, "M^4CD: A robust change detection method for intelligent visual surveillance," *IEEE Access*, vol. 6, pp. 15505–15520, 2018.

[5] T. Hirakawa *et al.*, "Tree-wise discriminative subtree selection for texture image labeling," *IEEE Access*, vol. 5, pp. 13617–13634, 2017.

[6] W. Song, D. Wu, Y. Xi, Y. W. Park, and K. Cho, "Motion-based skin region of interest detection with a real-time connected component labeling algorithm," *Multimedia Tools Appl.*, vol. 76, no. 9, pp. 11199–11214, 2017.

[7] J. Chen, Q. Gu, H. Gao, T. Aoyama, T. Takaki, and I. Ishii, "Fast 3-D shape measurement using blink-dot projection," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Nov. 2013, pp. 2683–2688.

[8] J. Chen, Q. Gu, T. Aoyama, T. Takaki, and I. Ishii, "Blink-spot projection method for fast three-dimensional shape measurement," *J. Robot. Mechatron.*, vol. 27, no. 4, pp. 430–443, 2015.

[9] P. Guler, D. Emeksiz, A. Temizel, M. Teke, and T. T. Temizel, "Real-time multi-camera video analytics system on GPU," *J. Real-Time Image Process.*, vol. 11, no. 3, pp. 457–472, 2016.

[10] L. He, X. Ren, Q. Gao, X. Zhao, B. Yao, and Y. Chao, "The connected-component labeling problem: A review of state-of-the-art algorithms," *Pattern Recognit.*, vol. 70, pp. 25–43, Oct. 2017.

[11] L. Cabaret, L. Lacassagne, and L. Oudni, "A review of world's fastest connected component labeling algorithms: Speed and energy estimation," in *Proc. Conf. Design Archit. Signal Image Process. (DASIP)*, Oct. 2014, pp. 1–6.

[12] L. He, Y. Chao, and K. Suzuki, "Two efficient label-equivalence-based connected-component labeling algorithms for 3-D binary images," *IEEE Trans. Image Process.*, vol. 20, no. 8, pp. 2122–2134, Aug. 2011.

[13] J. Martín-Herrero, "Hybrid object labelling in digital images," *Mach. Vis. Appl.*, vol. 18, no. 1, pp. 1–15, 2007.

[14] F. Chang, C.-J. Chen, and C.-J. Lu, "A linear-time component-labeling algorithm using contour tracing technique," *Comput. Vis. Image Understand.*, vol. 93, no. 2, pp. 206–220, 2004.

[15] L. He, Y. Chao, K. Suzuki, and K. Wu, "Fast connected-component labeling," *Pattern Recognit.*, vol. 42, no. 9, pp. 1977–1987, Sep. 2009.

[16] L. He, Y. Chao, and K. Suzuki, "A run-based two-scan labeling algorithm," *IEEE Trans. Image Process.*, vol. 17, no. 5, pp. 749–756, May 2008.

[17] C. Grana, D. Borghesani, and R. Cucchiara, "Optimized block-based connected components labeling with decision trees," *IEEE Trans. Image Process.*, vol. 19, no. 6, pp. 1596–1609, Jun. 2010.

[18] L. He, X. Zhao, Y. Chao, and K. Suzuki, "Configuration-transition-based connected-component labeling," *IEEE Trans. Image Process.*, vol. 23, no. 2, pp. 943–951, Feb. 2014.

[19] C. T. Johnston and D. G. Bailey, "Fpga implementation of a single pass connected components algorithm," in *Proc. 4th IEEE Int. Symp. Electron. Design, Test Appl. (DELTA)*, Jan. 2008, pp. 228–231.

[20] Q. Gu, T. Takaki, and I. Ishii, "Fast FPGA-based multiobject feature extraction," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 1, pp. 30–45, Jan. 2013.

[21] M. Manohar and H. K. Ramapriyan, "Connected component labeling of binary images on a mesh connected massively parallel processor," *Comput. Vis., Graph., Image Process.*, vol. 45, no. 2, pp. 133–149, 1989.

[22] R. Dewar and C. K. Harris, "Parallel computation of cluster properties: Application to 2D percolation," *J. Phys. A, Math. Gen.*, vol. 20, no. 4, p. 985, 1987.

[23] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," *Queue-GPU Comput.*, vol. 6, no. 2, pp. 40–53, Mar./Apr. 2008.

[24] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming, Portable Documents*. Reading, MA, USA: Addison-Wesley, 2010.

[25] K. A. Hawick, A. Leist, and D. P. Playne, "Parallel graph component labelling with GPUs and CUDA," *Parallel Comput.*, vol. 36, no. 12, pp. 655–678, Dec. 2010.

[26] I.-Y. Jung and C.-S. Jeong, "Parallel connected-component labeling algorithm for GPGPU applications," in *Proc. 10th Int. Symp. Commun. Inf. Technol. (ISCIT)*, Oct. 2010, pp. 1149–1153.

[27] O. Kalentev, A. Rai, S. Kemnitz, and R. Schneider, "Connected component labeling on a 2D grid using CUDA," *J. Parallel Distrib. Comput.*, vol. 71, no. 4, pp. 615–620, Apr. 2011.

[28] Y. Soh, H. Ashraf, Y. Hae, and I. Kim, "Fast parallel connected component labeling algorithms using CUDA based on 8-directional label selection," *Int. J. Latest Res. Sci. Technol.*, vol. 3, no. 2, pp. 187–190, 2014.

[29] S. Zavalishin, I. Safonov, Y. Bekhtin, and I. Kurilin, "Block equivalence algorithm for labeling 2D and 3D images on GPU," *Electron. Imag.*, vol. 2016, no. 2, pp. 1–7, 2016.
[30] O. Štàva *et al.*, "Connected component labeling in CUDA," in *GPU Computing Gems Emerald Edition*. Atlanta, GA, USA: Elsevier, 2011, pp. 569–581.
[31] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2009.
[32] V. M. A. Oliveira and R. A. Lotufo, "A study on connected components labeling algorithms using GPUs," in *Proc. SIBGRAPI*, vol. 3, 2010, p. 4.
[33] P. Kumar, A. Singhal, S. Mehta, and A. Mittal, "Real-time moving object detection algorithm on high-resolution videos using GPUs," *J. Real-Time Image Process.*, vol. 11, no. 1, pp. 93–109, 2016.
[34] J.-M. Park, C. G. Looney, and H.-C. Chen, "Fast connected component labeling algorithm using a divide and conquer technique," *Comput. Appl.*, vol. 4, pp. 4–7, Mar. 2000.
[35] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, "The floyd-warshall algorithm," in *Introduction to Algorithms*, vols. 558–565. Cambridge, MA, USA: MIT Press, 1990, pp. 570–576.
[36] K. Yonehara and K. Aizawa, "A line-based connected component labeling algorithm using GPUs," in *Proc. 3rd Int. Symp. Comput. Netw. (CANDAR)*, Dec. 2015, pp. 341–345.
[37] P. Chen, H. L. Zhao, C. Tao, and H. S. Sang, "Block-run-based connected component labelling algorithm for GPGPU using shared memory," *Electron. Lett.*, vol. 47, no. 24, pp. 1309–1311, Nov. 2011.
[38] H. L. Zhao, Y. B. Fan, T. X. Zhang, and H. S. Sang, "Stripe-based connected components labelling," *Electron. Lett.*, vol. 46, no. 21, pp. 1434–1436, Oct. 2010.
[39] F. N. Paravecino and D. Kaeli, "Accelerated connected component labeling using cuda framework," in *Computer Vision and Graphics*. Cham, Switzerland: Springer, 2014, pp. 502–509.
[40] Q. Xu, H. Jeon, and M. Annavaram, "Graph processing on GPUs: Where are the bottlenecks?" in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Oct. 2014, pp. 140–149.
[41] CUDA Nvidia, "Toolkit documentation v7. 0," Nvidia Corp., Santa Clara, CA, USA, Oct. 2016. [Online]. Available: http://docs.nvidia.com/cuda/index.html
[42] C. Grana, F. Bolelli, L. Baraldi, and R. Vezzani, "YACCLAB–Yet another connected components labeling benchmark," in *Proc. 23rd Int. Conf. Pattern Recognit. (ICPR)*, Dec. 2016, pp. 3109–3114.
[43] C. Grana, L. Baraldi, and F. Bolelli, "Optimized connected components labeling with pixel prediction," in *Advanced Concepts for Intelligent Vision Systems*. Cham, Switzerland: Springer, 2016, pp. 431–440.

JUN CHEN received the Ph.D. degree in system cybernetics from Hiroshima University, Japan, in 2015. From 2015 to 2016, he was a Post-Doctoral Researcher with the Robotic Laboratory, Hiroshima University, with a focus on high-speed image processing and sensing techniques. In 2016, he was a Research Engineer with KDDI Research, Inc., where his main focus is involved in the topics related to real-time image processing and analysis with some applications in free viewpoint video.

KEISUKE NONAKA received the Ph.D. degree in information processing from the Tokyo Institute of Technology, Japan, in 2014. He joined KDDI Research, Inc., in 2014, where he is currently a Research Engineer involving in image processing and computer vision. His research interests include free viewpoint video, virtual reality, and 3-D model rendering.

HIROSHI SANKOH received the B.E., M.E., and Ph.D. degrees in information science from Kyoto University in 2006, 2008, and 2015, respectively. He joined KDDI Research, Inc., in 2008, and since 2018, he has been engaged in the free viewpoint video technology field. He is currently a Research Engineer with the Ultra-Realistic Communication Group.

RYOSUKE WATANABE received the B.E. and M.E. degrees in information science from Hokkaido University in 2014 and 2016, respectively. He joined KDDI Research, Inc., Japan, in 2016. He is currently a Research Engineer with the Ultra-Realistic Communications Laboratory, KDDI Research, Inc.

HOUARI SABIRIN received the Ph.D. degree in information and communications engineering from KAIST, South Korea, in 2012. He is currently a Principal Research Engineer with KDDI Research, Inc., where his main focus is on topics related to object recognition and analysis with some applications in free viewpoint video.

SEI NAITO received the B.E., M.E., and Ph.D. degrees from Waseda University in 1994, 1996, and 2006, respectively. He joined KDDI Research, Inc., in 1996, where he is currently the Senior Manager of the Ultra-Realistic Communication Group. His research interests include free viewpoint video applications and virtual reality.