International Journal of New Computer Architectures and their Applications (IJNCAA) 6(4): 140-149
The Society of Digital Information and Wireless Communications, 2016 ISSN 2220-9085 (Online); ISSN 2412-3587 (Print)

GPU-based Object Identification in Large-scale Images for Real-time Radar Signal Analysis

Young-Min Kang
Tongmyong University
Busan, 48520, Korea
ymkang@tu.ac.kr

Sung-Soo Kim
ETRI
Daejeon, 34129, Korea
sungsoo@etri.re.kr

Gyung-Tae Nam
GCSC Inc.
Busan, 47607, Korea
gtnam@gcsc.co.kr

ABSTRACT

As the computing power of processors is being drastically improved, the sizes of image data for various applications are also increasing. However, CCL cannot be easily implemented in a parallel fashion because the connected pixels can be found basically only by graph traversal. In this paper, we propose a GPU-based efficient algorithm for object identification in large-scale images and the performance of the proposed method is compared with that of the most commonly used method implemented with OpenCV libraries. The experimental results show that the proposed method outperforms the reference method when the pixel density is below 0.7. Object identification in image data is the fundamental operation and rapid computation is highly requested as the sizes of the currently available image data rapidly increase. The method proposed in this paper is applied to the analysis of radar signals. Since the radar signals contains complex and large data, the efficiency of the method is crucial when realtime analysis is required. The experimental results show the proposed method can be a good solution to the object identification in large-scale image data such as radar signals.

KEYWORDS

GPGPU, Object Identification, CCL, Radar Signals, Realtime Analysis.

1 INTRODUCTION

As the computing power of processors is being drastically improved, the sizes of image data for various applications are also increasing. Therefore, efficient algorithms for manipulating the large-scale image data are required. One of the most basic operations on image data is to identify objects within the image, and the connected component labeling (CCL) is the most frequently used strategy for this problem.

Various image processing techniques can be easily implemented in parallel fashion, and GPU parallelism has been successfully exploited in this field. However, CCL cannot be easily implemented with parallel tasks because the connected pixels are represented as adjacent nodes in a graph and the adjacency among all the nodes can be investigated basically only by graph traversal.

In this paper, we propose a GPU-based efficient algorithm for object identification in large-scale images and the performance of the proposed method is compared with that of an OpenCV-based CCL algorithm.

2 RELATED WORK

Object identification is a fundamental problem in image processing. In many cases, the object identification is performed based on CCL. The most CCL algorithms are reduced to the traversal of adjacent nodes (pixels) along the edges in the graph that represents the input image. The traversal approaches are naturally sequential, and parallel implementations of typical CCL algorithms have not been very successful [10].

Even in the early stage of computer vision research, it was found that the connectivity cannot be determined by completely parallel tasks [7]. However, the rapid development of general purpose graphics processing unit (GPU) technologies made it possible for GPU-based parallel approaches to achieve better performance than CPU-based traditional CCL methods [1, 9].

The basic approach to CCL is to use *union-find* algorithm which can determine whether two nodes in an undirected graph are connected or not [8]. However, such methods based on this approach use sequential computations. Several methods have been proposed to exploit parallel computing architectures [5, 9]. However, these methods were applied to relatively small images. Some methods utilized cluster architectures [4]. These methods split the data volume and the data segments are assigned to different computing units. Parallelism within a single GPU, therefore, cannot be exploited in these methods.

Label equivalence method is implemented on GPU [6], and this method iterates to resolve the label equivalence by finding the roots of equivalence trees. However, this approach relies on decision tables which cannot be efficiently handled on GPU [10].

Block-based labeling and efficient block processing with decision table were proposed in [2, 3]. Block equivalence method based on "scan mask" was proposed [10]. However, this method also has to iterate the equivalence resolving until it converges to the state where no label update is found.

3 PROPOSED METHOD

In this section, a GPU-based parallel approach to CCL is proposed. The method is composed of four major tasks: 1) data initialization, 2) computation of column-wise label runs, 3) label merge of connected components. Each task is explained in details in the following subsections. The proposed method is to apply to radar signals. Therefore, the radar signals must be efficiently read and reconstructed in the format of normal images.

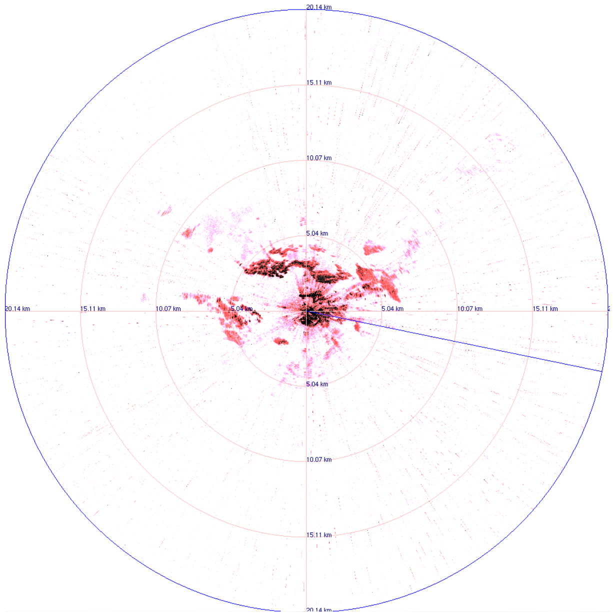


Figure 1. Radar signal visualization example

3.1 Real-time Radar Signal Visualization

The method proposed in this paper is to be applied to realtime analysis of large images such as radar signals shown in Fig. 1. ASTERIX is a well-known standard for the exchange of air traffic services (ATS) information developed by the European ATS organization Eurocontrol. In order to analyze the ASTERIX radar signal with image processing techniques, the signals must be parsed and reconstructed as rectangular image. The image reconstruction process is as follows:

1. Data Parsing: In this step, we read the radar signals in accordance with the data format. The reflectance intensity within the radar range is streamed into the system in azimuthal angle order.

2. Image Construction: The raw data in the radar signal is not in the form of rectangular images. In this step, we map the intensity signal into 4096×4096 image space.

3. Realtime Visualization: In order to visualize the radar signal mapped into 4096×4096 image in an efficient fashion, we allocate a buffer in the texture memory

to hold the image. The realtime visualization can then be easily implemented with GPU technologies.

3.2 Label Initialization

The pixels in an image can be classified into either 'on' pixels and 'off' pixels. The 'on' pixels are regarded nodes in graph representation, and it is assumed that an edge exists between two nodes of which positions neighbor each other in the image space.

The goal of CCL algorithms is to assign an identical label for linked nodes. In order to achieve this goal, each pixel is assigned unique label in the initialization stage. The simplest method is to assign sequential numbers to the pixels. Suppose we have an image with $w \times h$ pixels and the pixel at (x,y) is denoted as $p(x, y)$ where $x = [1, w], y = [1, h]$. The 'on' pixel at (x, y) is then labeled with the number $x + w(y - 1)$. Therefore, the labeling numbers range from 1 to wh. All the 'off' pixels are labeled to be -1.

The label map I_λ is an image composed of label of each pixel, and each label at (x, y) in the label map are denoted by $I_\lambda^{x,y}$. In other words, the image with initial labels can be described as follows:

$$
\begin{aligned}
I_\lambda &\in \mathbb{Z}^{w\times h} \\
id^{x,y} = x + w(y-1) &\quad \in [1, wh] \\
p(x,y) = 1 \Rightarrow l_\lambda^{x,y} &\quad = id^{x,y} \\
p(x,y) = 0 \Rightarrow l_\lambda^{x,y} &\quad = -1
\end{aligned} \tag{1}
$$

After the initialization is done, the rest of the algorithm is to merge the positive labels in each connected component into a single label.

3.3 Computing Column-wise Label Runs

In order to merge labels, adjacent positive labels are merged. The block of contiguous object pixels in a column is a 'run.' The first stage of label merge is to find runs. In other words, each run is identified and labeled with a unique number. Each column is assigned to a CUDA thread and processed in parallel fashion. Therefore, we have w threads running separately.

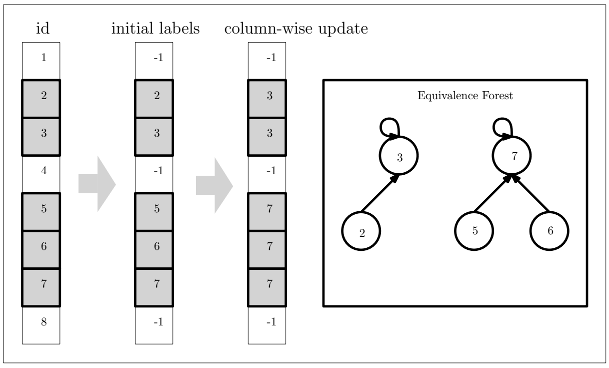


Figure 2. Label runs and equivalence forest

The computation within a thread is to simply scan pixels and change the label of the current pixel to be that of the previously scanned pixel if the both pixels are 'on.'

Fig. 2 shows how the label assigned to each pixel is updated through column-wise label run computation described in Algorithm 1. After the update, the each label represents the root node in the equivalence tree it belongs to as shown in Fig. 2

Algorithm 1: Column-wise label run

kernel **vertLabel**

Data: $I_\lambda \in \mathbb{Z}^{w\times h}$: In, Out
begin
  col = thread:$[1, w]$
  for *row:* $h - 1$ *downto* 1 **do**
    if $I_\lambda^{row,col} > 0$ *and* $I_\lambda^{row+1,col} > 0$ **then**
      $I_\lambda^{row,col} = I_\lambda^{row+1,col}$

3.4 Label Merge

Once the column-wise label update is finished, the horizontal connectivity must be investi-

International Journal of New Computer Architectures and their Applications (IJNCAA) 6(4): 140-149
The Society of Digital Information and Wireless Communications, 2016 ISSN 2220-9085 (Online); ISSN 2412-3587 (Print)

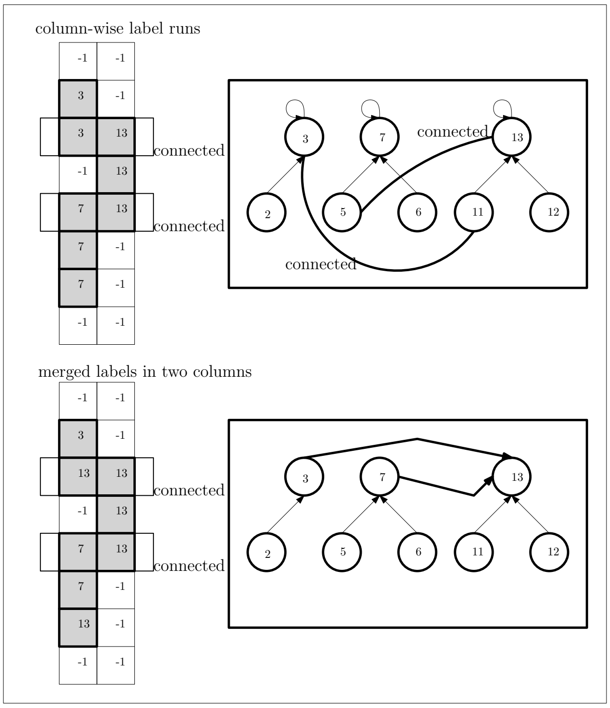


Figure 3. Label merge with two columns

gated. Let us suppose, for simplicity, that we have only two columns. In the previous column-wise update, the labels are merged to the largest value in the equivalence tree. If two pixels in a row are connected, the equivalence trees those pixels belong to should be merged into a single tree. In order to achieve this, the root node of each tree must be found and the root node with a smaller label is relabeled to point the other root with a larger label as shown in Fig. 3.

Note that the labels of the connected pixels are not directly updated. Fig. 4 shows the merge process. As shown in Fig. 4 (a), two pixels a and b in different equivalent trees are found to be connected. The direct update of connected pixels does not successfully merge the equivalence trees as shown in Fig. 4 (b). The correct label merge can be done by comparing and update of the roots of the equivalence trees as shown in Fig. 4 (c).

In order to perform the two-column label merge for an images with w columns, $w/2$ column pairs are separately merged with $w/2$ threads. After every two adjacent column pairs are merged, we iterate the label merge. In

Algorithm 2: Label merge algorithm

host **callMergeLabel**

Data: $I_\lambda \in \mathbb{Z}^{w \times h}$: In, Out
begin
  div = 2
  for *i:* 0 *upto* $\lg w - 1$ **do**
    merge $<<h \cdot w$/div$>>$ (div,I_λ)
    div = 2 $\times$ div

kernel **merge**

Data: div $\in \mathbb{Z}$: in, $I_\lambda \in \mathbb{Z}^{w \times h}$: In, Out
begin
  thread:$[1, wh$/div]
  nBoundary = w/div
  col = div/2 + (thread % nBoundary) $\times$ div
  row = thread / nBoundary
  if $I_\lambda^{row,col} > 0$ *and* $I_\lambda^{row,col+1} > 0$ **then**
    $root_L$ = **findRoot**(row,col)
    $root_R$ = **findRoot**(row,col+1)
    $I_\lambda^{\min(root_L,root_R)} = I_\lambda^{\max(root_L,root_R)}$

device **findRoot**

Data: row, col $\in \mathbb{Z}$: In, label $\in \mathbb{Z}$: Out
begin
  if $I_\lambda^{row,col} < 0$ **then**
    return -1
  label = w row + col
  while $I_\lambda^{label} \neq$ label **do**
    label := I_λ^{label}
  return label

Algorithm 3: Relabeling

kernel **relabel**

Data: $I_\lambda \in \mathbb{Z}^{w \times h}$: In, Out
begin
  thread:$[1, w \cdot h]$
  row = thread/w, col = thread%w
  $I_\lambda^{row,col} = \mathbf{findRoot}(row, col)$

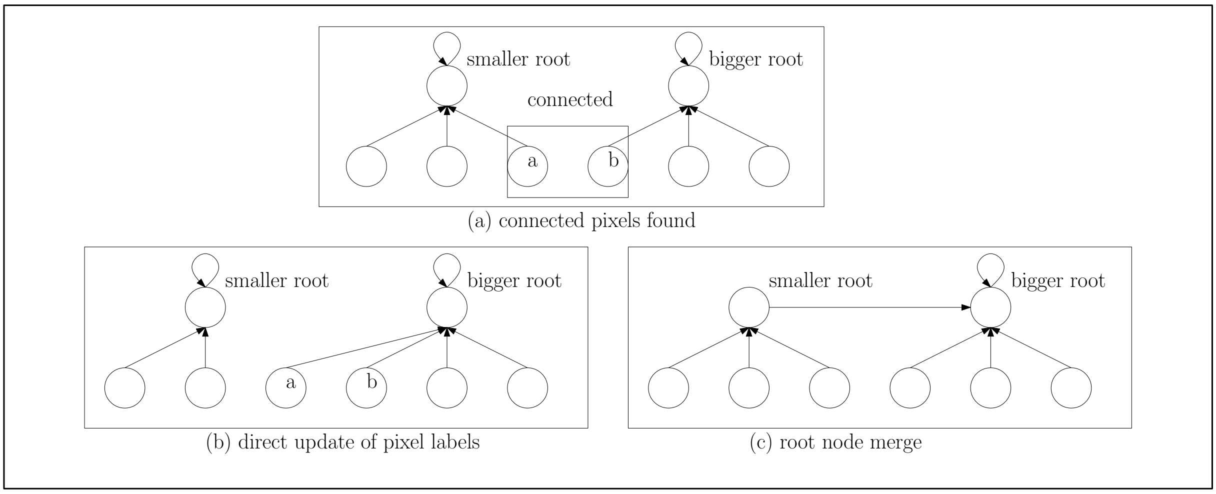


Figure 4. Label merge for equivalence tree merge

the second merge phase, as shown in Fig. 5, we have only to consider the boundaries between the previously merged column pairs, and the computation is reduced to half the previous one.

It is easily noticed that the $\lg w$ iterations are sufficient for finding the final label equivalence. Let us denote the computational cost for the first iteration to be $C(1)$. The total computational cost for the label merge is $\sum_{i=0}^{\lg w-1} \frac{1}{2^i} C(1) = \mathbf{O}(C(1))$.

Algorithm 2 shows the implementation details of our method. The label merge is implemented with one host function, one kernel function, and one device function. In the host function callMergeLabel, we determine the number of boundaries where column-pairs are merged and call kernel function merge with the necessary number of threads. The host function iterates this call $\lg w$ times, and the number of threads to be called decreases as the iteration is repeated. In the i=th call, $w \cdot h/2^i$ threads are required.

Every thread executes the kernel function merge. In the kernel function, every two pixels across the merge boundaries are investigated in parallel fashion. If the pixels are both 'on', the root nodes of equivalence trees the pixels belong to are found and compared. If they have identical root node, they are already in a single equivalence tree. Therefore, no more operations are required. Otherwise, they are in different equivalence trees. However, the horizontal adjacency of the pixels informs that they must be actually in an identical equivalence group. In this case, we must reassign the root labels of the pixels.

The reassign process can be understood as a tree merge as shown in Fig. 4. The label equivalence trees are merged by relabeling the root with smaller label to have the same label with the other root. The device function findRoot is called in this process to find the root of the pixel currently being investigated.

After the execution of Algorithm 2, the equivalence tree will be obtained. However, the final goal of CCL is to make all the pixels in a connected component have an identical label. This can be achieved by applying the device function findRoot to each pixel and updating its label to be the returned value. This process can be easily performed in a parallel fashion because the label updates for any nodes in the tree do not destroy the equivalence of the nodes in the tree.

Algorithm 3 describes the operations of relabeling thread. Total $w \cdot h$ threads separately call findRoot for corresponding pixels and update the label in order to make the connected pixels have an identical label.

International Journal of New Computer Architectures and their Applications (IJNCAA) 6(4): 140-149
The Society of Digital Information and Wireless Communications, 2016 ISSN 2220-9085 (Online); ISSN 2412-3587 (Print)

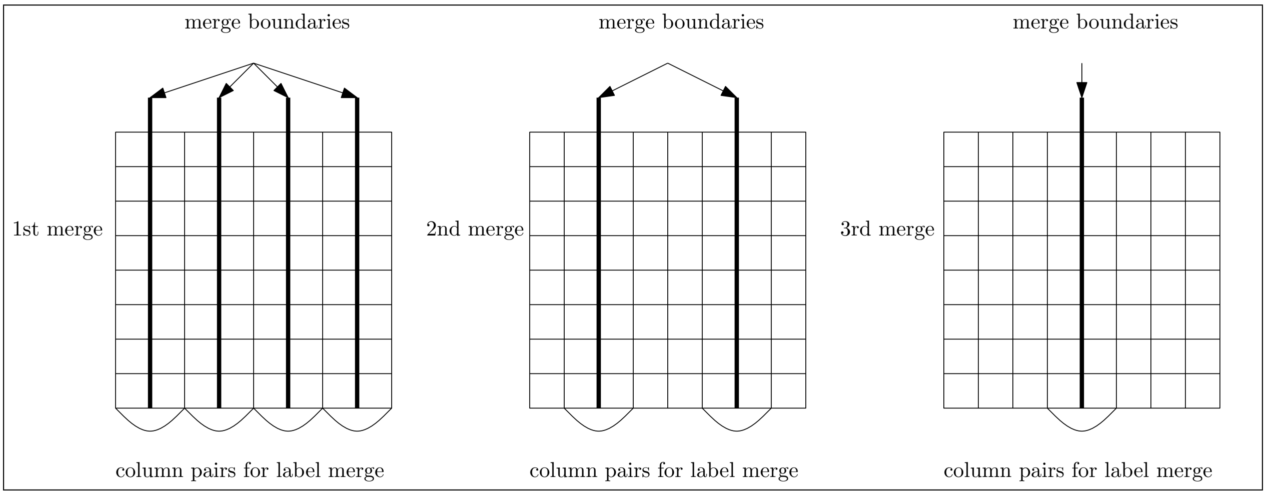


Figure 5. Label merge iteration

4 EXPERIMENTAL RESULTS

Table 1. CCL performance comparison with random density noise patterns (2048×2048 pixels).

2048×2048		
density	**Grana** (ms)	**Proposed method** (ms)
0.1	22.8	6.3
0.2	30.7	7.0
0.3	46.0	7.7
0.4	51.3	8.6
0.5	47.6	10.2
0.6	43.6	15.2
0.7	35.5	26.1
0.8	28,0	40.8
0.9	20.8	64.6

The method proposed in this paper was implemented on computing environments with commodity CPUs and GPUs. The experimental results were collected from the tests on a system with i7-3630QM 2.4 GHz CPU and Geforce GTX 670MX GPU.

In order to verify the efficiency of our method, we compared the performance of our method with the most commonly used method. The reference method was proposed in [3], and implemented with OpenCV libraries.

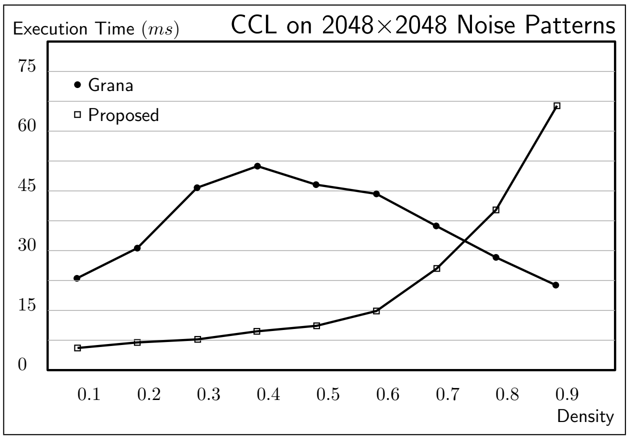
In the first experiment, the performance of each method was measured by applying images with random noise. The random noise was automatically generated and the density of the noise ranges from 0.1 to 0.9. Table. 1 shows the experimental results when 2048×2048 images with random noise are applied. The first column represents the noise density, and the second column shows the measured execution time of the reference method in milliseconds. The third column shows the execution time of the proposed method. As shown in the table, the reference method (denoted by Grana) requires more execution time when the noise density is around 0.5 while the proposed method requires more time as the density increases.

Table 2. CCL performance comparison with random density noise patterns (4096×4096 pixels).

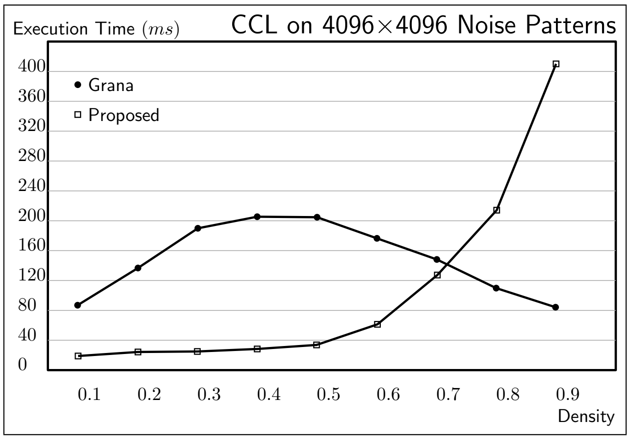
4096×4096		
density	**Grana** (ms)	**Proposed method** (ms)
0.1	85.7	21.4
0.2	135.3	24.3
0.3	190.4	27.1
0.4	206.1	30.8
0.5	205.4	36.9
0.6	177.6	59.0
0.7	153.1	126.7
0.8	112.1	216.7
0.9	85.6	411.5

Table. 2 shows the similar experimental results except that the size of the input images is 4096×4096. As shown in the table, the computational cost is similarly changing in accordance with the density.

Fig. 6 (a) visualizes the experimental results



(a)2048×2048-sized noise patterns



(b) 4096×4096-sized noise patterns

Figure 6. CCL execution time on noise patterns with different noise densities

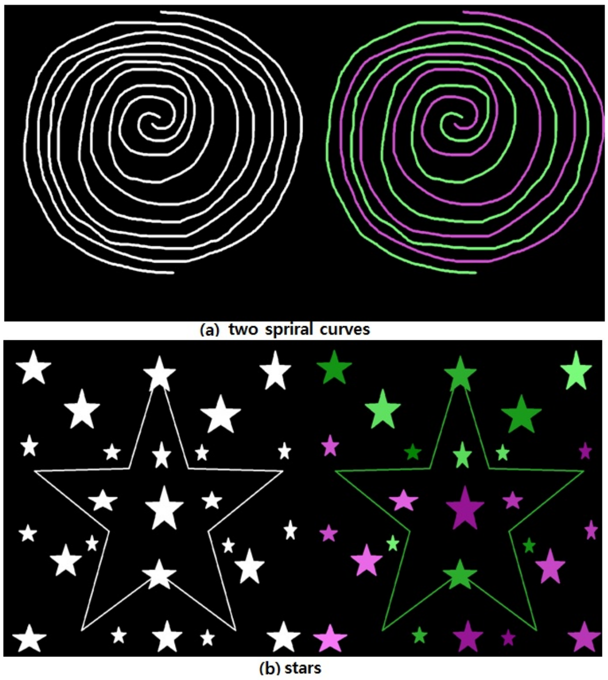


Figure 7. Test images for practical labeling: (a) two spiral curves (b) scattered stars

shown in Table. 1. As shown in the figure, the computational cost of the proposed method increases as the density increases. However, the proposed method is far better until the density is below 0.7.

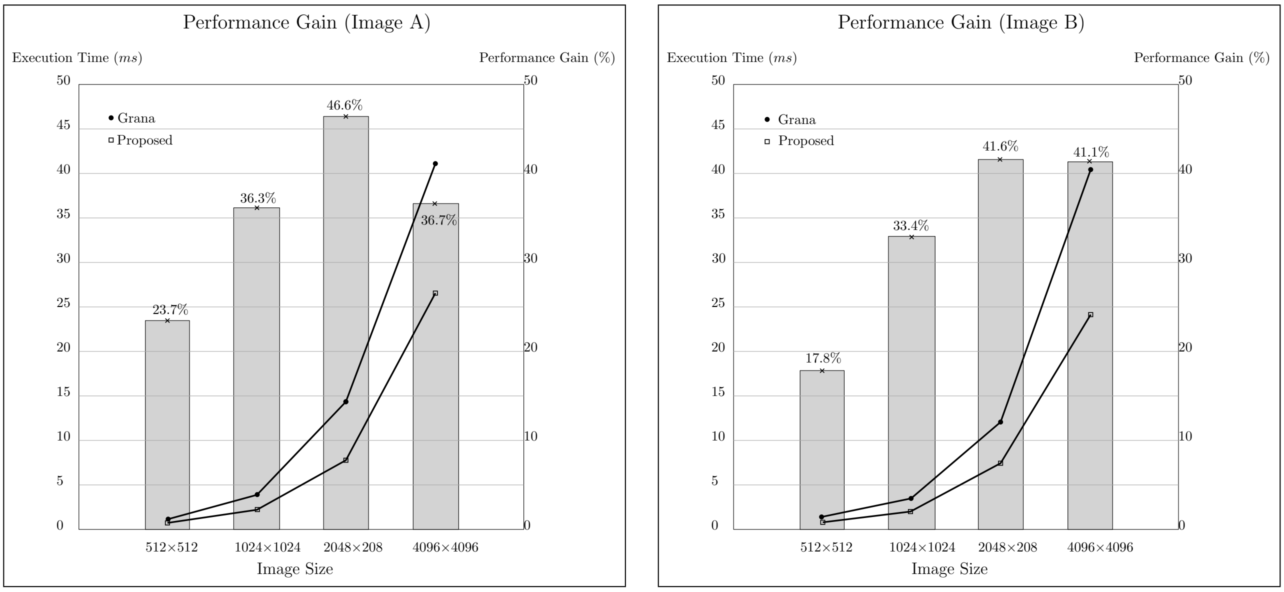
Fig. 6 (b) shows the similar results. This figure visualizes the result shown in Table. 2. As shown in the figure, the computational cost of the proposed method increases as the density increases. However, the proposed method is far better until the density is below 0.7.

Table 3. CCL performance comparison with test images A and B with different image sizes.

methods	Images sizes 512^2	1024^2	2048^2	4096^2
Test Image A				
Grana (ms)	1.14	3.86	14.39	41.1
Proposed (ms)	0.87	2.46	7.68	26.0
Gain (%)	23.7	36.3	46.6	36.7
Test Image B				
Grana (ms)	1.01	3.53	12.35	40.78
Proposed (ms)	0.83	2.35	7.21	24.03
Gain (%)	17.8	33.4	41.6	41.1

The CCL algorithms are not actually applied to noise data. In order to measure the performance of the proposed method in more feasible environments, we prepared two test images shown in Fig. 7. There are two different test images and the sizes of the images can be either 512^2, 1024^2, 2048^2 or 4096^2. Test image A has two components, and each of them is a long spiral curve without touching the other component. The other test image B has many scattered stars, and two of them are connected with a star-shaped thin line.

Table. 3 shows the execution time required for reference method and the proposed method applied to the test images with different sizes. In the last row in each data set obtained by using each test image, the performance gain is computed and shown. The performance gain is obtained by computing the ratio of the reduced computational cost by applying the proposed



(a)test image A (b)test image B

Figure 8. CCL execution time for test images and measured performance gains

Table 4. Performance analysis of GPU-based connected components labeling.

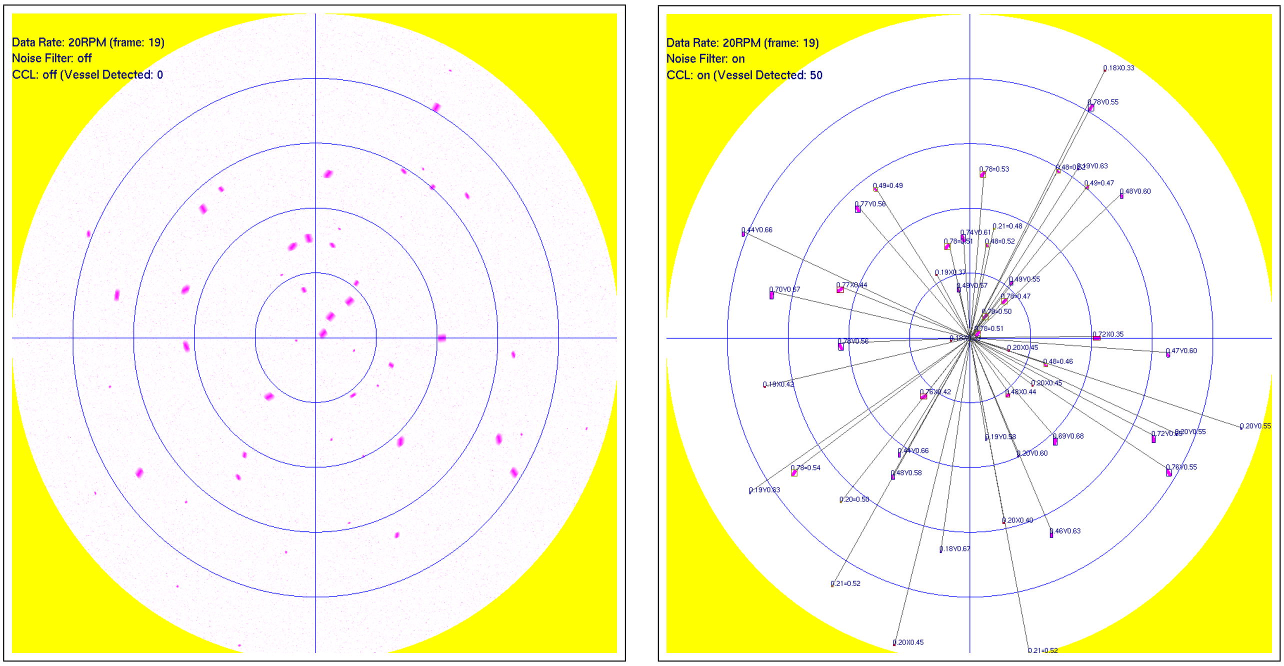
	Number of Connected Components	
Operations	50 connected components	1869 connected components
denoising	16.05 $msec$	16.05 $msec$
Preprocessing	16.05 $msec$	16.05 $msec$
label initialization	2.30$msec$	2.29$msec$
column-wise merge	3.59$msec$	3.63$msec$
hierarchical merge	7.97$msec$	8.35$msec$
relabel	3.07$msec$	3.17$msec$
CCL	16.93$msec$	17.44$msec$
counting components	2.12$msec$	2.16$msec$
bounding box computation	2.55$msec$	32.71$msec$
Bounding Box	4.67$msec$	34.87$msec$
Total Computation Time	21.60$msec$	52.31$msec$

method to the cost of the reference method. As shown in the table, the performance gain is more noticeable as the size of the input image increases.

Fig. 8 visually compares the computational costs of the reference method and our method (lines), and the performance gain is also visualized with bars. Fig. 8 (a) shows the result when the test image A was used as input image. The formance gain was the largest when the size of the input image is 2048×2048.

Fig. 8 (b) shows the similar results when the other test image is used. As shown in the figure, the performance gain is again the most noticeable when the size of the image is 2048×2048.

The method was applied to radar signal. In order to simulate the situation where a lot of moving vessels are detected by radar, we generated a radar signal as shown in Fig. 9 (a). The final images to be displayed has the size of 4096×4096. The input images were denoised and the proposed method was applied to those images. The connected components were successfully extracted as shown in Fig. 9 (b) in realtime. Fig. 10 (a) and (c) are zoom-in images of the result shown in Fig. 9 (a) and (b) respec-



(a) received radar signal (b) extracted connected commponents

Figure 9. The vessel tracking with proposed CCL techniques

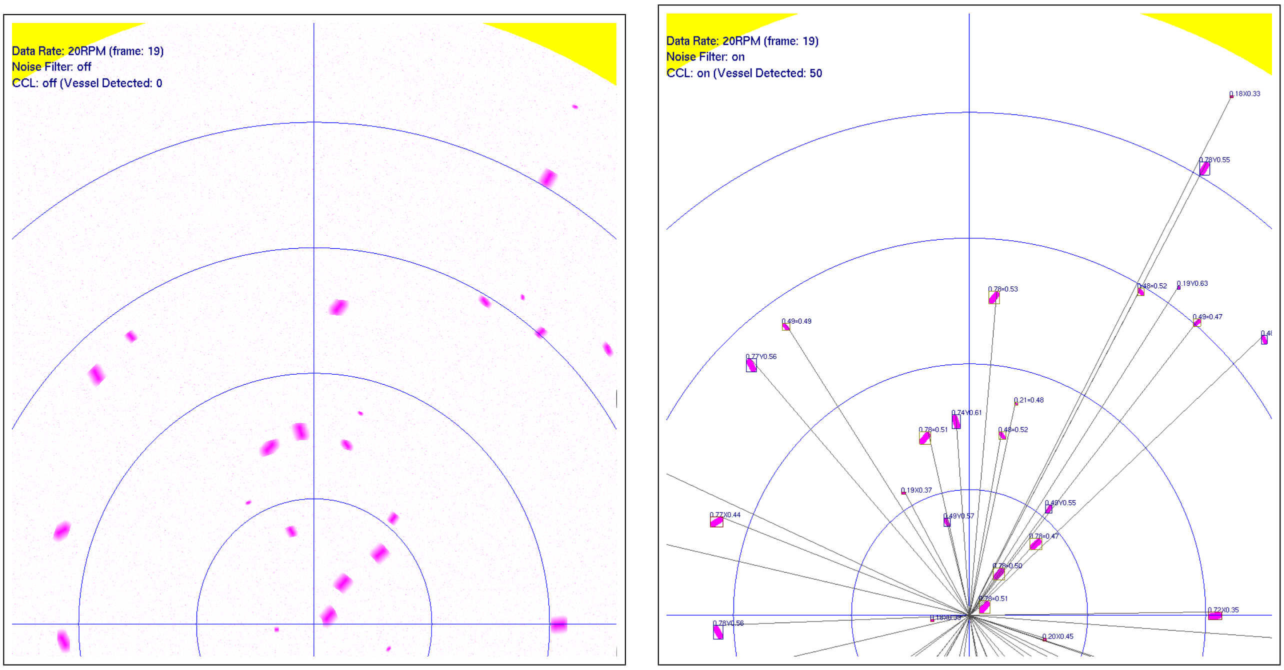
tively. As shown in the figure, the objects in the image were successfully labeled and tight bounding boxes wraping the components were obtained.

The performance analysis of GPU-based connected components labeling with radar signal is shown in Table 4. Each row shows the time required for each operations named in the first column in milliseconds. The operations can be categorized into three operation groups, and gray-colored rows shows the time consumed for those categories. We used two different radar signal data which have 50 and 1869 objects respectively.

As shown in the table, the computational costs of each operations do not rely on the number of connected components. Since the denoising process can be perfectly parallelized, the computational costs for both cases were exactly the same. Any preprocessing techniques can be applied in the phase, and the majority of image processing techniques can be easily parallelized. Even in the CCL process, any significant difference was not found in the aspect of computational costs. The CCL operation is composed of four subtasks. Each task required similar execution time for both cases. However, the bounding box computation required more time when we have more objects in the image because the generation of bounding boxes and determination of its properties requires computations of which complexity is proportional to the number of objects.

5 CONCLUSION

In this paper, an efficient GPGPU implementation of connected component labeling (CCL) was proposed. The method exploits the data parallelism of GPUs to improve the performance of CCL. Object identification in image data is the fundamental operation and rapid computation is highly requested as the sizes of the currently available image data rapidly increase. The experimental results show the proposed method can be a good solution to the object identification in large-scale image data. The CCL method proposed in this paper was successfully applied to realtime object tracking for radar signals. The computational cost of the proposed method does not hinder realtime visualization of radar signals.



(a) zoom-in image of input (b) zoom-in after CCL

Figure 10. The vessel tracking with proposed CCL techniques (Zoomed)

ACKNOWLEDGMENT

The research was supported by ‘Software Convergence Technology Development Program’, through the Ministry of Science, ICT and Future Planning (Development of Big Data Processing and Decision Support System for Offshore Maritime Safety, S0142-15-1014).

REFERENCES

[1] P. Chen, H. Zhao, C. Tao, and H. Sang. Block-run-based connected component labelling algorithm for gpgpu using shared memory. *Electronics letters*, 47(24):1309–1311, 2011.

[2] C. Grana, D. Borghesani, and R. Cucchiara. Connected component labeling techniques on modern architectures. In *International Conference on Image Analysis and Processing*, pages 816–824. Springer, 2009.

[3] C. Grana, D. Borghesani, and R. Cucchiara. Optimized block-based connected components labeling with decision trees. *IEEE Transactions on Image Processing*, 19(6):1596–1609, 2010.

[4] C. Harrison, H. Childs, and K. P. Gaither. Data-parallel mesh connected components labeling and analysis. In *Eurographics Parallel Graphics and Visualization Symposium, Llandudno, Wales*, 2012.

[5] K. A. Hawick, A. Leist, and D. P. Playne. Parallel graph component labelling with gpus and cuda. *Parallel Computing*, 36(12):655–678, 2010.

[6] O. Kalentev, A. Rai, S. Kemnitz, and R. Schneider. Connected component labeling on a 2d grid using cuda. *Journal of Parallel and Distributed Computing*, 71(4):615–620, 2011.

[7] M. Minsky and S. Papert. *Perceptrons*. MIT press, 1988.

[8] B. Preto, F. Birra, A. Lopes, and P. Medeiros. Object identification in binary tomographic images using gpgpus. *International Journal of Creative Interfaces and Computer Graphics (IJCICG)*, 4(2):40–56, 2013.

[9] O. Šťava and B. Beneš. Connected component labeling in cuda. *Hwu., WW (Ed.), GPU Computing Gems*, 2010.

[10] S. Zavalishin, I. Safonov, Y. Bekhtin, and I. Kurilin. Block equivalence algorithm for labeling 2d and 3d images on gpu. *Electronic Imaging*, 2016(2):1–7, 2016.