Parallel Connected Component Labeling Based on the Selective Four Directional Label Search Using CUDA

Young-Sung Soh*, ung-Woo Hong*

ABSTRACT

Connected component labeling (CCL) is a mandatory step in image segmentation where objects are extracted and uniquely labeled. CCL is a computationally expensive operation and thus is often done in parallel processing framework to reduce execution time. Various parallel CCL methods have been proposed in the literature. Among them are NSZ label equivalence (NSZ-LE) method, modified 8 directional label selection (M8DLS) method, HYBRID1 method, and HYBRID2 method. Soh et al. showed that HYBRID2 outperforms the others and is the best so far. In this paper we propose a new hybrid parallel CCL algorithm termed as HYBRID3 that combines selective four directional label search (S4DLS) with label backtracking (LB). We show that the average percentage speedup of the proposed over M8DLS is around 60% more than that of HYBRID2 over M8DLS for various kinds of images.

Keywords : Connected Component Labeling (CCL), Parallel Connected Component Labeling, Compute Unified Device Architecture (CUDA), Graphics Processing Unit (GPU)

I. Introduction

Connected component labeling (CCL) is a mandatory step in image segmentation where each object in an image is uniquely labeled. Many approaches were proposed in the field of CCL. Wu et al.[1] divided CCL methods into 3 categories. They are one-pass, two-pass, and multi-pass methods. One-pass algorithm scans the image only once and gives a new label to unlabeled pixel encountered during scanning. Then all the pixels connected to that pixel are searched recursively and are assigned the same label. Two-pass method consists of scanning, analysis, and labeling steps. Scanning step assigns an initial label to each pixel and keeps track of equivalence relations among labels, and analysis step determines a final label for each label by resolving equivalence relation chains. Finally labeling step assigns a final label to each intermediate label. Multi-pass method usually assumes some kind of local neighborhood to search for minimum label within that neighborhood and records label equivalence relations. This process is repeated until all the pixels are labeled using equivalence relations and no further change is detected in label assignment. Most of the methods used for CCL, irrespective of the number of passes they adopt, were sequential[2]. Since sequential CCL is time consuming and image resolution increases nowadays up to Full HD or UHD, it becomes very hard to process high resolution images in real time with CPU alone. To overcome this difficulty CCL has been implemented in parallel framework using graphics processing units (GPUs).

The usage of GPUs with compute unified device architecture (CUDA) developed by NVIDIA opened a new research era for the parallel implementation of CCL and many other data processing algorithms[3]. The parallel implementation of CCL using CUDA and GPUs drastically reduced computation time. Among many parallel CCL methods using CUDA proposed so far, HYBRID2[4] is known to perform best.

In this paper, we present a new hybrid method termed as HYBRID3 for parallel CCL using CUDA. We apply selective 4 directional label search (S4DLS) and label backtracking (LB)[4] alternately to initial label array. This alternation is repeated until there is no change in label assignment.

This paper is organized as follows. In section II, CUDA and related works for parallel CCL are presented. Section III describes the proposed method. Results are explained in section IV and section V gives the conclusion and discussion.

* 명지대학교
투고 일자 : 2015.5.19 수정완료일자 : 2015.7.22
게재확정일자 : 2015.8.8

Ⅱ. Related Works

2.1 CUDA

CUDA is the language used to implement parallel programs by utilizing GPUs on the graphics card produced by NVIDIA. CUDA programming model has a hierarchy as shown in Fig. 1.

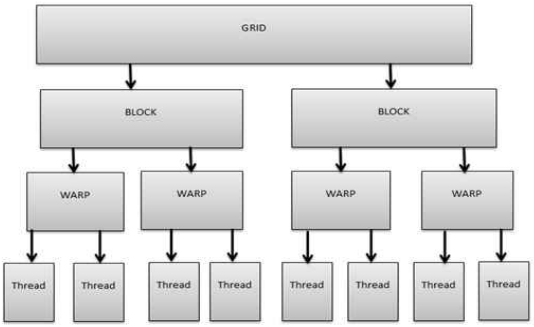


Fig. 1. CUDA programming model

Grid is at the top and threads are at the bottom. Grid has several blocks, each block has several warps, and each warp has several threads. Usually a single warp holds up to 32 threads. Blocks are also called cores or GPUs and all blocks run in parallel. All the threads in a block use shared memory. To do parallel processing, input image is fed from CPU to GPU global memory first. Depending on the number of cores selected for use, image in the GPU global memory is broken into sub images and they are fed into block's shared memories. After parallel processing is done in each block, results are combined to produce a single final output and can be transmitted to CPU memory if necessary.

2.2 CCL

Since 1980s many approaches have been proposed for the fast computation of CCL. Some of these approaches were sequential[1][2][5] and some others were parallel[4][6][7][8][9][10]. Since parallel CCL shows inherently faster performance than sequential version and the scope of this paper is confined to parallel CCL, we review relevant works pertaining only to parallel CCL.

Hawick et al.[9] proposed parallel version of the label equivalence algorithm using GPUs. Their algorithm consists of scanning, analysis and labeling steps. These steps are repeated in a loop until all the object pixels are labeled correctly. Since their method is implemented within parallel framework, the execution time was greatly reduced. However it consumes more memory space in using the reference array.

Kalentov et al.[6] developed two parallel methods. The first one is a simple row column unification method where each row and column is assigned a single thread. Each thread scans corresponding row or column in a predefined direction, and changes the label of each pixel with the minimum label found so far along the scan direction. This process is repeated until all pixels are changed with minimum labels. The second method is the NSZ-LE where each pixel is assigned a single thread. Each thread searches for minimum label in immediate 4 neighbors, keeps track of the label equivalence chain, and performs relabeling by resolving equivalence chains.

Soh et al.[7] proposed 8 directional label equivalence (8DLS) method where the minimum label is searched in both immediate and further 8 neighbors rather than only 4 immediate neighbors of the focused pixel until background pixel is hit. Each object pixel is assigned a single thread and all the threads run in parallel. This process is repeated until all object pixels have minimum labels.

Soh et al.[7] developed M8DLS which is an improved version of 8DLS. They reduce the search space by not processing the focused pixel if it already has minimum label, thus saving execution time. Determination of minimum label is done by checking if the original position of that label still has that label.

Soh et al.[8] proposed HYBRID1 method where M8DLS and MKC[9] are applied in alternate fashion to increase the efficiency. It was shown that HYBRID1 performs better than M8DLS.

Soh et al.[4] proposed HYBRID2 method where M8DLS and LB are processed at alternate iterations to increase the efficiency more than HYBRID1. In comparison with HYBRID1 the execution time was greatly reduced by adopting label backtracking rather than deep search.

Ⅲ. The Proposed Method

In this chapter, we propose a new hybrid parallel CCL method termed as HYBRID3 which is a combination of S4DLS and LB[4]. First we present S4DLS and LB in detail and then describe HYBRID3.

3.1 S4DLS

Unlike M8DLS[7] where we search all 8 directions for label search, S4DLS breaks 8 directions into two 4 directional groups such that one group has smaller labels than focused pixel's label and the other group has bigger labels. The former and latter groups are termed as UPPER GROUP and LOWER GROUP respectively. The comparison of M8DLS and S4DLS along with the location of two groups is shown in Fig. 2(a) and 2(b) respectively. The numbers from 0 to 7 represent search directions. M8DLS searches for all 8 directions whereas S4DLS searches UPPER GROUP first and if neighbors are found, then LOWER GROUP is not searched. An example of S4DLS is illustrated in Fig. 2(c). Fig. 2(c) shows initial label array where white and shaded pixels represent object and background pixels respectively. The pixel with label 22 has neighbors in UPPER GROUP (indicated by 3 black arrows), thus searching LOWER GROUP is not necessary. However the pixel with label 35 does not have neighbors in UPPER GROUP, therefore searching LOWER GROUP is required (indicated by 1 black arrow).

For ordinary images, the number of object pixels having a neighborhood only in LOWER GROUP is relatively much lower comparing to the total number of object pixels. Thus the computation time will be greatly saved.

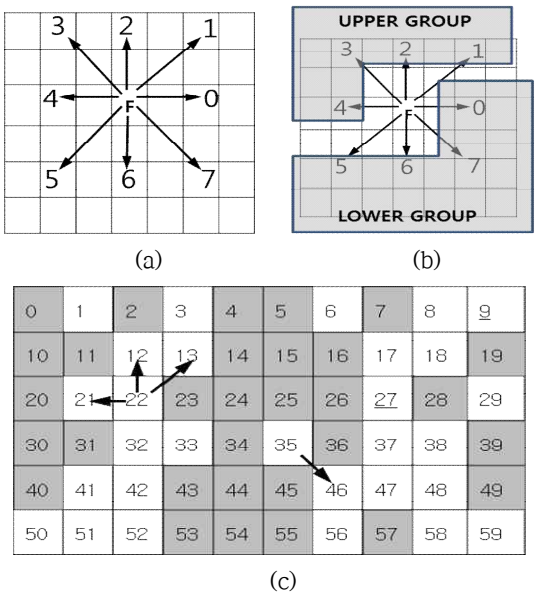


Fig. 2. Comparison of M8DLS and S4DLS. (a) M8DLS, (b) S4DLS, (c) example of S4DLS

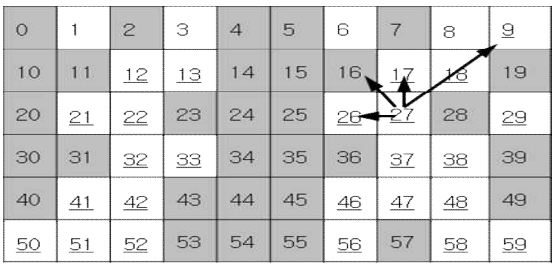
The pseudo code for S4DLS is given in Algorithm 1.

Algorithm 1. The S4DLS method in pseudo code

```
for each pixel p in an image do
    if p is an object pixel
    then  it becomes a focused pixel
          for each direction in upper group do
              search for minimum label until
background pixel is hit
              and put it in min_i
          end for
          if no minimum found in upper group
          then for each direction in lower group do
                   search for minimum label until
background pixel is hit
                   and put it in min_i
               end for
          end if
      end if
      take minimum label m among min_i , 1<= i <= 8
 and relabel the focused pixel with label m
  end for
```

In S4DLS, 4 directional search in UPPER GROUP (directions 1, 2, 3, and 4 in Fig. 2(b)) is performed for each object pixel (focused pixel). Search continues until background pixel is hit. While searching, minimum label is selected for each direction, thus obtaining maximum 4 intermediate minima. Final minimum is the minimum of 4 possible intermediate minima. The label of focused pixel is changed with the final minimum. If focused pixel has no neighbors in UPPER GROUP, other 4 directions in LOWER GROUP (directions 5, 6, 7, and 0 in Fig. 2(b)) will be searched and final minimum of those 4 directions will be a final minimum. If minima was found in UPPER GROUP, then we skip directions in LOWER GROUP, thus searching time will be reduced in half. In Fig. 3, we show a running example of S4DLS. Fig. 3(a) is a sample input image that has been uniquely and sequentially labeled according to the indices of the pixels in the image. Here white pixel is an object pixel and the shaded is a background pixel. Assuming 8-connectivity, there are two objects. Each object pixel is assigned a single thread and the algorithm is executed only on object pixels. Fig. 3(b), 3(c), and 3(d) depict the results obtained after first, second, and third iterations respectively. To see how it works, let us consider the pixel with label 27 underlined in Fig. 3(a). We search for 4 directions in UPPER GROUP (indicated by 4 black arrows which extend to either background pixel or the boundary) and find that the label 9 (underlined in Fig. 3(a)) in 45° diagonal direction is the minimum. Thus the label 27 is changed to 9 as underlined in Fig. 3(b) at the same location. Since there were

neighbors in UPPER GROUP, LOWER GROUP is not searched. In Fig. 3(a), object pixels with underlined label indicate that only UPPER GROUP search was conducted for those pixels. As was expected, we can see that most of the object pixels needs only 4 directional search in UPPER GROUP, thus saving search time endeavoring LOWER GROUP. For this example 3 iterations were enough to correctly label all the object pixels.



(a)

0	1	2	3	4	5	6	7	8	8
10	11	1	3	14	15	16	6	8	19
20	3	12	23	24	25	8	9	28	18
30	31	12	22	34	35	36	17	27	39
40	32	12	43	44	45	37	17	26	49
32	33	12	53	54	55	29	57	38	26

(b)

0	1	2	1	4	5	6	7	6	8
10	11	1	1	14	15	16	6	6	19
20	1	1	23	24	25	6	6	28	18
30	31	1	12	34	35	36	6	9	39
40	12	1	43	44	45	17	6	8	49
12	12	1	53	54	55	17	57	17	8

(c)

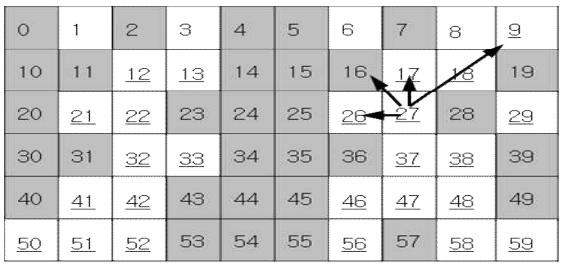
0	1	2	1	4	5	6	7	6	6
10	11	1	1	14	15	16	6	6	19
20	1	1	23	24	25	6	6	28	6
30	31	1	1	34	35	36	6	6	39
40	1	1	43	44	45	6	6	6	49
1	1	1	53	54	55	6	57	6	6

(d)

Fig. 3. Running example of S4DLS method. (a) initial label, (b) after first iteration, (c) after second iteration, and (d) after third iteration

3.2 LB

We adopt to use label backtracking (LB) proposed in [4] and it is performed as follows. Let LABEL(p) be the label of focused pixel p under consideration. Then we backtrack to LABEL(LABEL(p)) and check if the condition LABEL(LABEL(p)) == LABEL(p) is met. If it is, then we stop, otherwise we further backtrack to LABEL(LABEL(LABEL(p))) and check if the condition LABEL(LABEL(LABEL(p))) == LABEL(LABEL(p)) is met. If it is, then we stop, otherwise we keep backtracking until the condition is met. A running example of LB is depicted in Fig. 4. As in Fig. 3(a), Fig. 4(a) is an initial label array and Fig. 4(b) is the result after executing S4DLS. We apply LB to Fig. 4(b) to get Fig. 4(c). Let us consider the pixel labeled 27 (underlined in Fig. 4(b)) as the focused pixel. Since the label is 27, we backtrack to the location which initially has label 27. The location is underlined in Fig. 4(a). In that location, we found that the label is 9 (underlined in Fig. 4(b)). Since LABEL(27) != 27 in Fig. 4(b), we backtrack to the location with initial label 9 (underlined in Fig. 4(a)). There we found that the label is 8 (underlined in Fig. 4(b)). Since LABEL(9) != 9 in Fig. 4(b), we backtrack to the location with initial label 8. In that location, the label is 8 (marked with parentheses in Fig. 4(b)) and the condition LABEL(8) == 8 is met. Thus the label 27 of the focused pixel is finally changed to 8 (underlined in Fig. 4(c)) and LB processing for the focused pixel is completed. The arrows depicted in Fig. 4(b) show the label backtracking route.



(a)

(b)

0	1	2	3	4	5	6	7	8	8
10	11	1	3	14	15	16	6	8	19
20	3	1	23	24	25	8	8	28	8
30	31	1	1	34	35	36	6	8	39
40	1	1	43	44	45	6	6	8	49
1	1	1	53	54	55	8	57	8	8

(c)

Fig. 4. Running example of LB method. (a) initial label, (b) after applying S4DLS, and (c) after applying LB

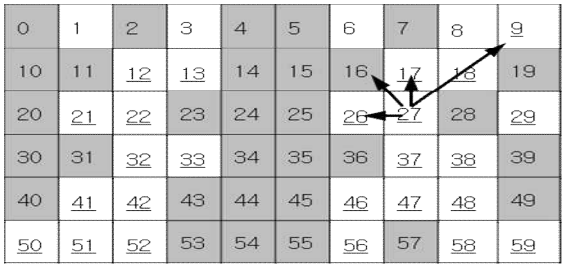
3.3 HYBRID3

HYBRID3 method is described in Algorithm 2. S4DLS and LB are applied in alternate fashion such that S4DLS and LB are applied at odd and even iterations respectively.

Algorithm 2. The HYBRID3 method in pseudo code

```
for i=1 to n iterations do
  for each pixel p in an image do
    if p is an object pixel
    then it becomes a focused pixel
         if i % 2 == 1
         then apply S4DLS
         else apply LB
         end if
    end if
  end for
 if no label change for all the object pixels
 then exit
 end if
 end for
```

Fig. 5 shows the running example of HYBRID3. Fig. 5(a) is the initial label. Fig. 5(b), 5(c), 5(d), and 5(e) are results after first, second, third, and fourth iteration respectively. S4DLS was applied at first and third iterations, and LB was applied at second and fourth iterations. In Fig. 5(a), object pixels with underlined label indicate that only UPPER GROUP search was conducted for those pixels. As was expected, most of the object pixels needs only 4 directional search in UPPER GROUP, thus saving search time endeavoring LOWER GROUP. In Fig. 5(e), two objects are labeled correctly with labels 1 and 6 that are minimum labels for two objects.



(a)

0	1	2	3	4	5	6	7	8	8
10	11	1	3	14	15	16	6	8	19
20	3	12	23	24	25	8	9	28	18
30	31	12	22	34	35	36	17	27	39
40	32	12	43	44	45	37	17	26	49
32	33	12	53	54	55	29	57	38	26

(b)

0	1	2	3	4	5	6	7	8	8
10	11	1	3	14	15	16	6	8	19
20	3	1	23	24	25	8	8	28	8
30	31	1	1	34	35	36	6	8	39
40	1	1	43	44	45	6	6	8	49
1	1	1	53	54	55	8	57	8	8

(c)

0	1	2	1	4	5	6	7	6	8
10	11	1	1	14	15	16	6	6	19
20	1	1	23	24	25	6	6	28	8
30	31	1	1	34	35	36	6	6	39
40	1	1	43	44	45	6	6	6	49
1	1	1	53	54	55	6	57	6	6

(d)

0	1	2	1	4	5	6	7	6	6
10	11	1	1	14	15	16	6	6	19
20	1	1	23	24	25	6	6	28	6
30	31	1	1	34	35	36	6	6	39
40	1	1	43	44	45	6	6	6	49
1	1	1	53	54	55	6	57	6	6

(e)

Fig. 5. Running example of HYBRID3 method. (a) initial label, (b) after applying S4DLS, (c) after applying LB, (d) after applying S4DLS, and (e) after applying LB

Ⅳ. Experimental Results

The system specification used for the experiment is as follows.

- **CPU**: Intel i7-4770, 3.40 GHz
- **OS**: Windows 7
- **GPU**: NVIDIA Geforce GTX 550 Ti with 192 cores.

For the experiment, 8 types of images were used. All are of size 320 x 240 with 8 bits/pixel. They are,

- **B1, B2, B3, B4, and B5**: Binary images with occupancy ratio of 0.07, 0.17, 0.27, 0.36 and 0.46 respectively
- **Spiral**: Binary image with one big spiral pattern
- **Random1 and Random2**: Binary images that were produced by scattering object pixels at the locations generated by random number generator and have the occupancy ratio of 0.1 and 0.5 respectively.

We compared the performance of M8DLS[7], HYBRID1[8], HYBRID2[4], and the proposed method (HYBRID3) in terms of execution time. The comparison with NSZ-LE[6] is also added for reference. Table 1 describes the comparison results of execution time. We run each algorithm 100 times on each test image and take the average execution time. Regardless of the methods tested, when we go from B1 to B5, execution time increases due to increasing occupancies. The same observation can be made when we go from Random1 to Random2. We can observe that NSZ-LE is the worst and computation time decreases as we go from M8DLS to HYBRID1, HYBRID2 and then to HYBRID3. The data for NSZ-LE, M8DLS, HYBRID1, and HYBRID2 shown in Table 1 is somewhat different from those in Table 1 in [4] due to the fact that different CPU, mother board, and RAM are used.

Table 1. Comparison of execution time (unit: msec)

Images	NSZ-LE [6]	M8DLS [7]	HYBRID1 [8]	HYBRID2 [4]	HYBRID3
B1	12.34	7.10	6.89	6.64	6.51
B2	20.41	7.38	7.11	6.86	6.58
B3	21.69	7.76	7.37	7.02	6.60
B4	22.31	8.06	7.77	7.28	6.63
B5	24.51	8.26	8.17	7.52	6.65
Spiral	35.37	8.09	7.67	7.27	6.61
Random1	10.12	7.37	7.18	6.71	6.50
Random2	15.23	7.73	7.53	6.93	6.53

Table 2 shows the speedup percentages of HYBRID1, HYBRID2, and HYBRID3 over M8DLS. In average, HYBRID1 obtains 3.3% speedup over M8DLS, while HYBRID2 and HYBRID3 achieve 9.0% and 14.6% speedup respectively. The average speedup percentage of HYBRID3 over M8DLS is around 60% more than that of HYBRID2 over M8DLS. Though the actual increase in speedup percentage from HYBRID2 to HYBRID3 is 5.6%P, it is a meaningful improvement since HYBRID2 is known to be the fastest and can be considered as almost an optimized version of parallel CCL using CUDA.

This speedup was achieved since, for most of the object pixels, we look at directions only in UPPER GROUP.

Table 2. Comparison of speedup over M8DLS (unit: %)

Images	HYBRID1 [8]	HYBRID2 [4]	The Proposed (HYBRID3)
B1	3.0	6.5	8.3
B2	3.7	7.0	10.8
B3	5.0	9.5	14.9
B4	3.6	9.7	17.7
B5	1.1	9.0	19.5
Spiral	5.2	10.1	18.3
Random1	2.6	9.0	11.8
Random2	2.6	10.3	15.5
Average Speedup	3.3	9.0	14.6

Ⅴ. Conclusion

CCL is a necessary step in image segmentation and is often implemented in parallel framework to reduce execution time. Soh et al. proposed HYBRID1 that is the combination of M8DLS and MKC, and showed a slight improvement over previous methods. Soh et al. also proposed HYBRID2 that combines M8DLS with LB and reported far better performance than HYBRID1. In this paper, we proposed HYBRID3 that uses S4DLS and LB in alternate fashion and showed that HYBRID3 performed better than HYBRID2 for various kinds of images.

VI. Reference

[1] K. Wu, E. Otoo, and K. Suzuki, "Optimizing two-pass connected-component labeling algorithms," Pattern Analysis & Applications vol. 12 issue 2, pp. 117-135, 2009.

[2] K. Suzuki, I. Horiba, and N. Sugie, "Linear-time connected-component labeling based on sequential local operations," Computer Vision and Image Understanding vol. 89 issue 1, pp. 1-23, 2003.

[3] R. Farber, CUDA Application Design and Development, Waltham: Elsevier, 2011.

[4] Y. Soh,H. Ashraf, and I. Kim, "An Improved Hybrid Approach to Parallel Connected Component Labeling using CUDA," Journal of the Institute of Signal Processing and Systems, Vol. 16, No. 1, pp.1-8, Jan. 2015.

[5] F. Chang, C. Chen, and C. Lu, "A linear-time Component-labeling algorithm using contour tracing technique," Computer Vision and Image Understanding vol. 93 issue 2, pp. 206-220, 2004.

[6] O. Kalentov, A. Rai, S. Kemnitz, and R. Schneider, "Connected component labeling on a 2D grid using CUDA," J. Parallel Distributed Computing vol. 71, pp. 615-620, 2011.

[7] Y. Soh, H. Ashraf, Y. Hae and I. Kim, "Fast Parallel Connected component labeling Algorithm in CUDA based on 8-Directional Label Selection," International Journal of Latest Research in Science and Technology, pp. 187-190, 2014.

[8] Y. Soh, H. Ashraf, Y. Hae and I. Kim, "A Hybrid Approach to Parallel Connected Component Labeling Using CUDA," International Journal of Signal Processing Systems, Vol. 1, No. 2, pp. 130-135, 2013.

[9] K. Hawick, A. Leist, and D. Playne, "Parallel graph component labeling with GPUs and CUDA," Parallel Computing vol. 36 issue 12, 2010.

[10] Y. Soh, H. Ashraf, Y. Hae and I. Kim," A Simple and Fast parallel Connected Component Labeling using CUDA, " in Proceedings of International Conference on Computer Applications and Information Processing Technology, pp. 61-64, 2013.

Young-Sung Soh

(Regular member)

Received BS in electrical engineering in 1978 from Seoul National University in Seoul, Korea, and MS and PhD in computer science from the University of South Carolina in Columbia, South Carolina, USA in 1986 and 1989, respectively.

He is currently a professor in the Dept. of Information and Communication Engineering, Myongji University, Korea. His current interest of research includes object tracking, stereo vision, and parallel algorithms for image processing.

Jung-Woo Hong

Received BS in Information and Communication engineering in 2012 from Myongji University in Kyunggido, Korea, and is currently in MS program in the Dept. of Information and Communication Engineering at Myongji University.

His current interest of research includes CUDA-based parallel algorithms for image processing.