An Improved Hybrid Approach to Parallel Connected Component Labeling using CUDA

Young-Sung Soh*, Hadi Ashraf*, and In-Taek Kim*

ABSTRACT

In many image processing tasks, connected component labeling (CCL) is performed to extract regions of interest. CCL was usually done in a sequential fashion when image resolution was relatively low and there are small number of input channels. As image resolution gets higher up to HD or Full HD and as the number of input channels increases, sequential CCL is too time-consuming to be used in real time applications. To cope with this situation, parallel CCL framework was introduced where multiple cores are utilized simultaneously. Several parallel CCL methods have been proposed in the literature. Among them are NSZ label equivalence (NSZ-LE) method[1], modified 8 directional label selection (M8DLS) method[2], and HYBRID1 method[3]. Soh [3] showed that HYBRID1 outperforms NSZ-LE and M8DLS, and argued that HYBRID1 is by far the best. In this paper we propose an improved hybrid parallel CCL algorithm termed as HYBRID2 that hybridizes M8DLS with label backtracking (LB) and show that it runs around 20% faster than HYBRID1 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

In many image processing tasks, connected component labeling (CCL) is performed to extract regions of interest. Various approaches were proposed in the field of CCL. Wu[9] divided CCL methods into 3 categories. They are one-pass, two-pass, and multi-pass methods. One-pass algorithm scans the image from top-left to bottom-right just 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. This recursive label assignment is performed until all pixels are labeled.

Two-pass method consists of 3 steps. They are scanning, analysis, and labeling steps. Scanning step assigns an intial label to each pixel and keeps track of equivalence relations among labels if necessary while searching for neighbors.

Analysis step determines a final label for each lavel by resolving equivalence relation chains. Finally labeling step assigns a final label to each 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, regardless of the number of passes they adopt, were sequential[4][10]. Sequential CCL can be used successfully in a single CPU system that processes a small number of channels of image stream with relatively low resolution. Though sequential CCL is a computationally expensive operation, increasing power of CPU enables sequential CCL run in real time for a few input channels with relatively low resolution. However, as the number of channels increases, say up to 32 or 64 and as image resolution increases up to HD or Full HD, it becomes very hard to process all high resolution input streams in real time with CPU alone. To overcome this difficulty it has been tried to implement CCL in parallel framework using graphics processing units (GPUs).

The usage of GPUs with compute unified device architecture (CUDA) developed by NVIDIA opened a new

* 명지대학교 정보통신공학과
투고 일자 : 2014. 12. 29 수정완료일자 : 2015. 1. 26
게재확정일자 : 2015. 2. 2

research era for the parallel implementation of CCL and many other data processing algorithms[5]. The parallel implementation of CCL using CUDA and GPUs drastically reduced computation time. Among many parallel CCL methods using CUDA proposed so far, HYBRID1[3] is known to perform best.

In this paper, we present an improved hybrid method termed as HYBRID2 for parallel CCL using CUDA. We first give a unique and consecutive label to every pixel in the image to produce initial label array. Next we apply M8DLS[2] and label backtracking (LB) alternately. This alternation is repeated until there is no change in label assignment. M8DLS searches for minimum label of each object pixel in 8 directions (east, west, south, north, and 4 diagonal directions) until background pixel is hit and changes the label of that pixel with minimum label found. In LB, for the label L of each object pixel, we backtrack to the location L in initial label array and check if it has a label L or not. If it does, we stop backtracking. Otherwise we continue backtracking to the location indicated by the label of previous backtracking location. The proposed algorithm will be described in detail in later section.

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

II. Related Works

2.1 CUDA

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

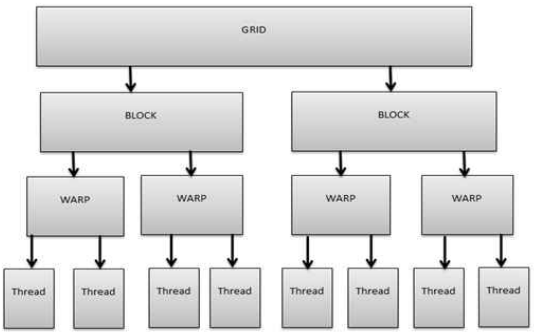


Fig. 1. CUDA programming model

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 uo 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[6][9][10] and some others were parallel[1][2][3][7][8]. Suzuki[10] proposed a simple sequential CCL suitable for the implementation in hardware. However it is not fast enough since the execution time of their method grows proportionally with the number of pixels in connected components in an image, hence making it not suitable for images with high resolution.

Wu[9] proposed the scan based array union-find algorithm which is basically an optimized version of traditional two-pass algorithm. This algorithm performs 10 times better than contour tracing algorithm[6] and other previous methods[10] in accessing the memory pattern in CCL. However it suffers from the loss of efficiency when images with small resolution are processed.

Chang[6] developed the contour tracing algorithm which has a better computational speed than [9] and [10] but consumes more time in accessing the memory pattern in CCL.

Hawick[7] proposed parallel version of the label equivalence algorithm using GPUs. As in two-pass algorithm, their algorithm consists of three basic steps: scanning, analysis and labeling. 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 in using the reference array.

Kalentov[1] 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[8] 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[2] 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.

Soh[3] proposed HYBRID1 method where M8DLS and modified kernel C (MKC)[7] are applied in alternate fashion to increase the efficiency. It was shown that HYBRID1 performs better than M8DLS.

Ⅲ. The Proposed Method

In this chapter, we propose a new parallel CCL method termed as HYBRID2 which is a combination of M8DLS and label backtracking (LB). First we present M8DLS and LB in detail and then describe HYBRID2.

3.1 M8DLS

M8DLS[2] makes use of 8DLS[8]. The pseudo code for the M8DLS method is given in Algorithm 1.

Algorithm 1 The M8DLS 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) and (label of p is not the smallest)
            then apply 8DLS
            end if
    end if
  end for
 if no label change for all the object pixels
 then exit
 end if
 end for
```

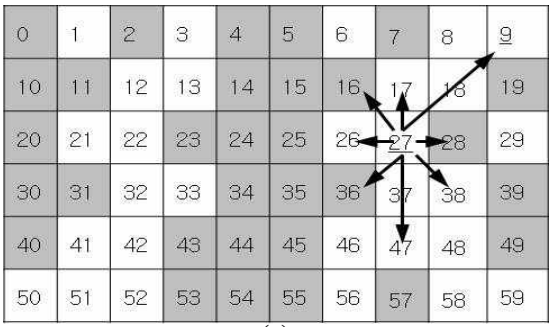
In Algorithm 1, 8DLS is applied if some conditions are met. The pseudo code for the 8DLS algorithm is given in Algorithm 2 [8].

Algorithm 2 The 8DLS 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 i = 1 to 8 directions do
                search for minimum label until
background pixel is hit
                and put it in mini
            end for
     end if
     take minimum label m among min_i , 1<= i <= 8
 and relabel the focused pixel with label m
  end for
```

In 8DLS, 8 directional search(east, west, south, north, and 4 diagonal directions) 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 8 intermediate minima. Final minimum is the minimum of 8 intermediate minima. The label of focused pixel is changed with the final minimum. M8DLS is the modification of 8DLS in that, after second iteration, the label of the focused pixel is checked if it is the smallest so far. If it is not, 8DLS is applied. Otherwise, no further processing is performed until next checkup, thus saving computation time. Checking for smallest is performed as follows. Let LABEL be an initial label image array. Then after assigning initial label sequentially to an image, LABEL(i) becomes i, for i = 0 to (total number of pixels in the image) - 1 as in Fig. 2(a). For focused pixel p having a label j, we check if LABEL(j) is still j. If it is, we say that j is the smallest label so far. Otherwise it is not the smallest since it has already been changed, thus having a possibility of further change. If LABEL(j) is changed to something else later, then we apply 8DLS again to pixels having label j. Algorithm halts when there is no label change for all the object pixels in the image. The running appearance of M8DLS for initial label array is exactly same as that of 8DLS since smallest checking is done from second iteration. However, many pixels will not be processed while producing the same results in later iterations. In Fig. 2, we show a running example of M8DLS. Fig. 2(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. 2(b), 2(c), and 2(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. 2(a). We search for 8 directions (indicated by 8 black arrows which extend to either background pixel or the boundary) and find that the label 9 (underlined in Fig. 2(a)) in 45° diagonal direction is the minimum. Thus the label 27 is changed to 9 as underlined in Fig. 2(b) at the same location. 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. 2.. Running example of M8DLS method. (a) initial label, (b) after first iteration, (c) after second iteration, and (d) after third iteration

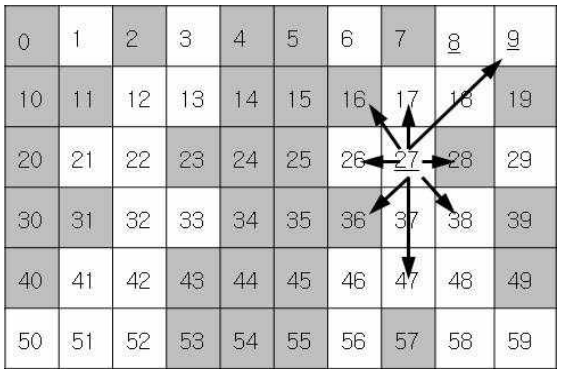
In Fig. 2(c), we put the underline for object pixels that pass the "smallest" check described above. For left and right objects, 9 out of 13 and 8 out of 16 pixels were not processed respectively, thus saving great amount of computation time.

In M8DLS, "smallest" label check is performed after two iterations. This number was chosen empirically. We conducted many experiments for various numbers of iterations and for various kinds of test data, and found that, after two iterations, most of object pixels already has smallest label they ought to have due to deep 8-directional search characteristic of the method.

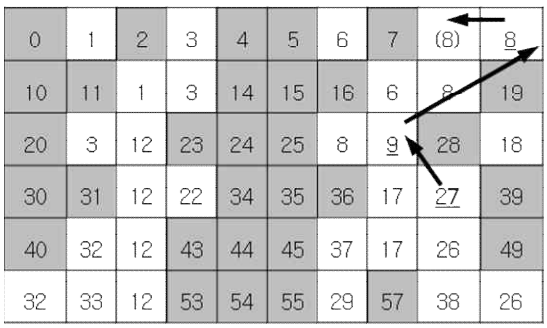
3.2 LB

Label backtracking (LB) 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. 3. As in Fig. 2(a), Fig. 3(a) is an initial label array and Fig. 3(b) is the result after executing M8DLS. We apply LB to Fig. 3(b) to get Fig. 3(c). Let us consider the pixel labeled 27 (underlined in Fig. 3(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. 3(a). In that location, we found that the label is 9 (underlined in Fig. 3(b)). Since LABEL(27) != 27 in Fig. 3(b), we backtrack to the location with initial label 9 (underlined in Fig. 3(a)). There we found that the label is 8 (underlined in Fig. 3(b)). Since LABEL(9) != 9 in Fig. 3(b), we backtrack to the location with initial label 8. In that location, the label is 8 (marked

within parenthesis in Fig. 3(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. 3(c)) and LB processing for the focused pixel is completed. The arrows depicted in Fig. 3(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. 3. Running example of LB method. (a) initial label, (b) after applying M8DLS, and (c) after applying LB

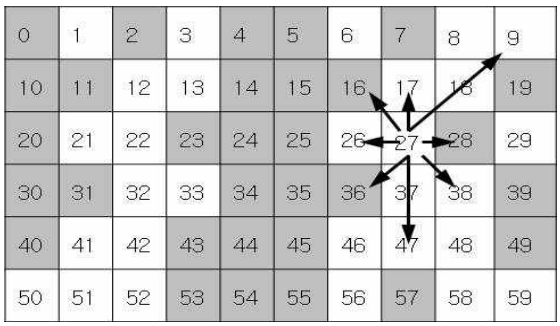
3.3 HYBRID2

HYBRID2 method is described in Algorithm 3. M8DLS and LB are applied in alternate fashion. M8DLS and LB are applied at odd and even iterations respectively.

Algorithm 3 The HYBRID2 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 M8DLS
        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. 4 shows the running example of HYBRID2. Fig 4(a) is the initial label. Fig 4(b), 4(c), 4(d), and 4(e) are results after first, second, third, and fourth iteration respectively. M8DLS was applied at first and third iterations, and LB was applied at second and fourth iterations. In Fig. 4(e), we can see that 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. 4. Running example of HYBRID2 method. (a) initial label, (b) after applying M8DLS, (c) after applying LB, (d) after applying M8DLS, and (e) after applying LB

Ⅳ. Experimental Results

The system specification used for the experiment is as follows.

- **CPU**: Intel i7, 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 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[2], HYBRID1 [3], and the proposed method (HYBRID2) in terms of execution time. The comparison with NSZ-LE[1] 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. Irrespective 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 HYBRID2 takes a lot less computation time than M8DLS and HYBRID1.

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

Images	NSZ-LE	M8DLS	HYBRID1	The Proposed (HYBRID2)
B1	26	7.93	7.72	6.29
B2	28	8.21	8.13	6.45
B3	30	8.51	8.32	6.57
B4	31	8.74	8.58	6.79
B5	34	9.13	8.99	6.83
Spiral	135	9.90	8.87	6.63
Random1	29.7	8.03	7.82	6.37
Random2	30.4	8.13	7.96	6.47

Table 2 shows the speedup percentage of HYBRID1 and HYBRID2 over M8DLS. HYBRID1 performs slightly better than M8DLS, whereas HYBRID2 shows far better performance. In average, HYBRID1 obtains 3.0% speedup over M8DLS, while HYBRID2 achieves 23.3% speedup. This speedup was achieved since, in LB, we look at only necessary locations of label array by backtracking instead of searching many other locations as in conventional methods.

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

Images	HYBRID1	The Proposed (HYBRID2)
B1	2.6	20.7
B2	1.0	21.4
B3	2.2	22.8
B4	1.8	22.3
B5	1.5	25.2
Spiral	10.4	33.0
Random1	2.6	20.7
Random2	2.1	20.4
Average Speedup	3.0	23.3

Ⅴ. Conclusion

CCL is a necessary step in image segmentation and is often implemented in parallel framework to reduce execution time. Soh[2] compared NSZ-LE[1] with 8DLS[8] and M8DLS, and showed that both 8DLS and M8DLS outperformed NSZ-LE and M8DLS performs better than 8DLS. Soh[3] also proposed HYBRID1 that is the combination of M8DLS and MKC, and showed a slight improvement than previous methods. In this paper, we proposed an improved hybrid parallel CCL method, termed as HYBRID2, where we combine M8DLS with LB. We showed that HYBRID2 outperforms all other conventional methods for various kinds of images.

Reference

[1] O. Kalentev, 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

[2] 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

[3] 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

[4] A. Rosenfeld and A. Kak, Digital Picture Processing, Orlando: Academic Press, 1982

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

[6] 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

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

[8] 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.

[9] 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

[10] 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

Young-Sung Soh

(Regular member)

received BS in electrical engineering in 1978 from Seoul National University and MS and PhD in computer science from the University of South Carolina 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.

Hadi Ashraf

received BS in electrical engineering in 2010 from Govt. University in Lahore, Pakistan and MS in information and communication engineering from Myongji University, Korea in 2014.

His current interest of research includes object tracking, stereo vision and parallel algorithms for image processing.

In-Taek Kim

received BS and MS in electronics engineering from Seoul National University in 1980 and 1984 respectively, and PhD in electrical engineering from Georgia Institute of Technology in 1992.

He is now a professor in the Dept. of Information and Communication Engineering, Myongji University. His research interest includes pattern recognition, image processing and smart grid area.