# Parallel Connected-Component Labeling Algorithm for GPGPU Applications

In-Yong Jung[*] and Chang-Sung Jeong[†]
[*]Korea University, Republic of Korea
E-mail: dekarno@korea.ac.kr Tel: +82-02-921-0471
[†]Korea University, Republic of Korea
E-mail: csjeong@korea.ac.kr Tel: +82-02-3290-3229

*Abstract*— **This paper proposes a new connected component labeling algorithm for GPGPU applications based on NVIDIA's CUDA. Various approaches and algorithms for connected component labeling with minimal execution time were designed, but the most of them have been focused on optimizing CPU algorithm. Therefore it is hard to apply these approaches to GPGPU programming models such as NVIDIA's CUDA. Today, GPGPU (General Purpose Graphic Processing Unit) technologies offer dedicated parallel hardware and programming model, and many applications are being moved onto the GPGPU. This algorithm is a multi-pass algorithm to utilize for GPGPU applications, and evaluation results show that maximum speedup is more than double compared with conventional CPU algorithms.**

## I. Introduction

Connected component labeling is the operation to transform a binary image into a symbolic image in which all connected components are assigned a unique label. In computer vision, labeling of connected component is one of the basic and most fundamental operations when a system needs to recognize objects or patterns in input images. Therefore, lower execution time and complexity are required of connected component labeling algorithms to apply to real time systems.

Many approaches and algorithms for addressing this labeling problem have been proposed, but many of them are focusing on sequential approaches and their optimizations for ordinary computer architecture [8]. Therefore, it is hard to apply these sequential approaches to GPGPU programming models such as NVIDIA's CUDA (Compute Unified Device Architecture) [1].

GPGPU (General Purpose Graphic Processing Unit) technologies offer dedicated data parallel hardware with SIMD architecture and unique parallel programming models and languages such as CUDA or ATI Stream [2]. This technologies make programmers are available to access GPGPU hardware directly and accelerate their applications with massive data parallelism. Today, many scientific applications are being moved onto the GPU.

In this paper, we introduce novel GPGPU connected component labeling algorithm based on NVIDIA's CUDA. This algorithm is a multi-pass algorithm suitable for GPGPU applications, and performance evaluations demonstrate that our algorithm is very competitive as compared with conventional sequential connected component labeling algorithms.

The rest of this paper is organized as follows: we classify conventional connected component labeling approaches in the next section, and we introduce our algorithm in section 3. In section 4, we show performance evaluation result. A short conclusion is given in section 5.

## II. Conventional Connected-Component Labeling Algorithms

In papers by Suzuki, Wu et al. [3-5], approaches and algorithms for connected component labeling are categorized into a number of groups as follows.

*A.* *Multi-Pass Algorithms* scan an image in the forward and backward raster directions alternately to propagate label equivalences until no label changes.

*B.* *Two-Pass Algorithms* operate in three phases; Scanning, Analysis and Labeling. In generally, these methods show very high performance, but require large memory to store label equivalence. The performance of these algorithms is very dependent on complexity of connected component and speed of resolving the label equivalences at Analysis phase. There is a type of hybrid [3] between multi-pass and two-pass algorithm. Like multi-scan algorithms, the hybrid algorithm scans an image in the forward and backward raster directions alternately. During the scans, as in two-scan algorithms, a one-dimensional table is used for recording and resolving label equivalences.

*C.* *One Pass Algorithms* need not to analyze label equivalences by tracing the contours of objects or by use of an iterative recursion operation.

Additionally, there are parallel algorithms designed for special device architectures such as a mesh connected massively parallel processors, GPGPU and CBE [7-9], and hardware implementation of this type has been studied. According to a paper described by Hawick et al. [8], four GPGPU connected component labeling algorithms based on CUDA have been proposed for hypercubic mesh graphs. GPGPU programming model with unique hardware architecture requires different approach as compared to CPU algorithms and optimal memory usage is important issue for GPGPU application. We have implemented and evaluated a new connected component labeling algorithm on GPGPU based on CUDA.

## III. Fast Connected-Component Labeling Algorithm on GPGPU

The algorithm proposed in this paper has 6 phases and requires a number of iterations to complete labeling operations. Therefore it can be regarded as a kind of multi-pass algorithm. In the GPGPU with CUDA architecture, program can access and handle multiple pixel data by using unique indices assigned to each thread and grid block. In this algorithm, threads are allocated to each image pixel to do labeling operation. In addition, this algorithm requires a 1 or 2 dimensional array with same size of the input image. This array, called Label Array, takes a role of equivalence list and fully labeled output image. Our algorithm flow is shown in Fig. 1.

### *A. Init Phase*

The algorithm proposed in this paper has 6 phases and requires a number of iterations to complete labeling operations. Therefore it can be regarded as a kind of multi-pass algorithm. Init phase assigns initial labels of each pixel belonging to objects. If a pixel is belonging to object, unique index of each thread (allocated to each pixel) is assigned to index of the pixel in the Label Array. Otherwise, background value (Vb) is assigned. Thread operations with background pixels will be ignored in following phases. See Fig. 4(b).

### *B. Scanning Phase*

In Scanning phase, each thread examines labels of directly neighboring pixels (A, B, C, D) belonging to a mask shown in Fig. 2, introduced as 'forward scan mask' in [4-5]. After that, threads find and write lowest label including itself to the Label Array. Because initially assigned labels from lower index to higher index are increasing value, threads need not to scan neighborhood pixels excluding the mask. If there are no neighborhood pixels in the mask, label of the pixel takes a role of representative label, and maintains same label in Label Array. After this phase, Label Array will have label equivalence information of all pixels. See Fig. 4(c).

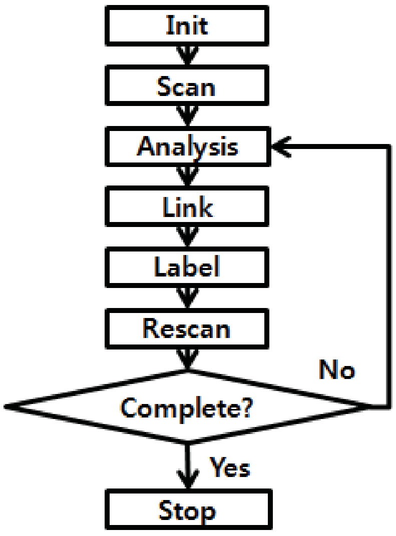


Fig. 1 Flow of Proposed Algorithm

### *C. Analysis Phase*

In Analysis phase, each thread find representative label as a root of each pixel to propagate it to each sub-region by using labels written in Label Array. Representative label means a label written in Label Array equal to each pixel index. If threads find each representative label, threads write them into Label Array. After this phase, Label Array will have a number of sub-regions, labeled with same representative labels. See Fig. 4(d). In the case Fig. 4(d), there are 3 sub-regions labeled with 1, 3 and 21.

### *D. Link Phase*

This phase links each connected sub-region to build a fully labeled connected component. Each thread examine labels of neighborhood pixels (C, D, F, G) belonging to a mask shown in Fig. 3 and find lowest label, similar to the Scanning phase. If a pixel in a sub-region is directly neighboring with other sub-regions, lower or higher representative labels can be found. In these cases, if a found label is lower than representative label of thread, thread will update label of its representative label index. See Fig. 4(e).

In this phase, congestion caused by heavy thread race is expected, but it can be resolved by using atomicMin function in CUDA. When a thread does memory operation with atomic function, no other thread can access the occupied address until the operation is completed [1].

### *E. Label Phase*

In Label phase, each thread finds its representative label to its pixel. After this phase, Label Array will have a number of large sub-regions or fully labeled connected components if all directly connected sub-regions are linked. See Fig. 4(f).

### *F. Rescan Phase*

After Label phase, each thread scans labels of all neighborhood pixels to check all directly connected sub-regions are linked. If component labeling is not completed, Rescan phase set a flag to true, then host will repeat execution of Analysis, Link, Label, and Rescan phase in order. The

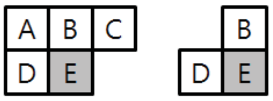


Fig. 2 Scanning Mask for $N_8$(left), $N_4$(right)

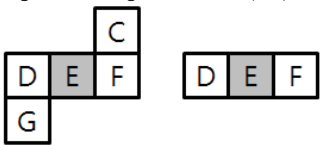
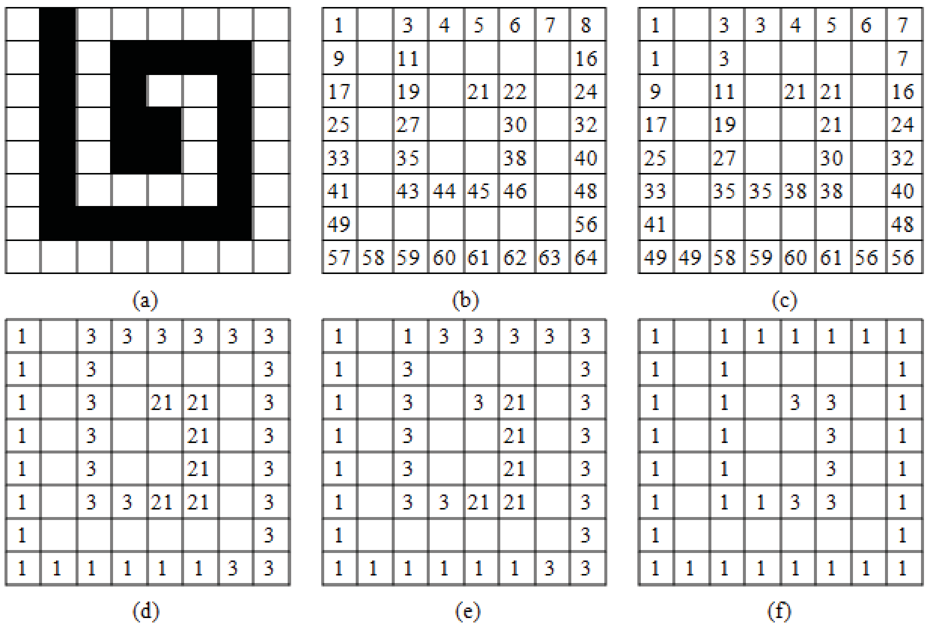


Fig. 3 Link Scanning Mask for $N_8$(left), $N_4$(right)



(b)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | | 11 | | | | | 16 |
| 17 | | 19 | | 21 | 22 | | 24 |
| 25 | | 27 | | | 30 | | 32 |
| 33 | | 35 | | | 38 | | 40 |
| 41 | | 43 | 44 | 45 | 46 | | 48 |
| 49 | | | | | | | 56 |
| 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 |

(c)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | | 3 | 3 | 4 | 5 | 6 | 7 |
| 1 | | 3 | | | | | 7 |
| 9 | | 11 | | 21 | 21 | | 16 |
| 17 | | 19 | | | 21 | | 24 |
| 25 | | 27 | | | 30 | | 32 |
| 33 | | 35 | 35 | 38 | 38 | | 40 |
| 41 | | | | | | | 48 |
| 49 | 49 | 58 | 59 | 60 | 61 | 56 | 56 |

(d)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | | 3 | 3 | 3 | 3 | 3 | 3 |
| 1 | | 3 | | | | | 3 |
| 1 | | 3 | | 21 | 21 | | 3 |
| 1 | | 3 | | | 21 | | 3 |
| 1 | | 3 | | | 21 | | 3 |
| 1 | | 3 | 3 | 21 | 21 | | 3 |
| 1 | | | | | | | 3 |
| 1 | 1 | 1 | 1 | 1 | 1 | 3 | 3 |

(e)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | | 1 | 3 | 3 | 3 | 3 | 3 |
| 1 | | 3 | | | | | 3 |
| 1 | | 3 | | 3 | 21 | | 3 |
| 1 | | 3 | | | 21 | | 3 |
| 1 | | 3 | | | 21 | | 3 |
| 1 | | 3 | 3 | 21 | 21 | | 3 |
| 1 | | | | | | | 3 |
| 1 | 1 | 1 | 1 | 1 | 1 | 3 | 3 |

(f)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | | 1 | | | | | 1 |
| 1 | | 1 | | 3 | 3 | | 1 |
| 1 | | 1 | | | 3 | | 1 |
| 1 | | 1 | | | 3 | | 1 |
| 1 | | 1 | 1 | 3 | 3 | | 1 |
| 1 | | | | | | | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Fig. 4 Results of each phase in Label Array: (a) Input Image; (b) Init; (c) Scan; (d) Analysis; (e) Link; (f) Label

results in Label Array after each phase are shown in Fig. 4.

## IV. Experiments & Results

The performance of the proposed algorithm has been tested with two different image sets: Noise Image Set and Realistic Image Set. Noise image set contains 312 noise images with various sizes (from 128x128 to 1024x1024) and densities (from 2.5% to 97.5%). Each noise image was generated by thresholding of the images containing uniform random noise to build complex connected components. Realistic image set consists of thresholded 298 binary images obtained from the USC Image Database [10], Ground Truth Database of the University of Washington [11] and FVC2000 databases [12]. In addition to this image set, 16 binary spiral images with various sizes (from 128x128 to 2048x2048) were included. See Fig. 5.

In our experiments, comparative evaluations of the labeling execution time were performed. Time required at data transfer between CPU and GPGPU was excepted from the execution time measurement for GPGPU. We compared our GPGPU algorithm with two conventional labeling algorithms proposed in following references: (1) the hybrid CCL algorithm (CPU1) in Ref. [3]; (2) the linear-time two-scan CCL algorithm (CPU2) in Ref. [6].

All algorithms used for testing were implemented in C language including CUDA syntax, and compiled under the same condition. All experimental results were obtained by averaging of the execution time for 100 runs on a PC with Intel Q6600 Core2Quad 2.66 GHz Processor, 4GB Memory, NVIDIA Geforce GTX260 GPGPU and Windows 7 32bit OS.

### A. *Algorithm Implementation*

Each phase (Init, Scan, Analysis, Link, Label, Rescan phase) described in previous chapter is implemented as a device kernel of CUDA and executed and repeated in order. Only the global memory of GPGPU is used in this implementation. This algorithm requires more phases than conventional sequential CCL algorithms and requires several iterations for object images with complex connectivity. However, required iterations are just a few, and overall execution time is less than expectation. In a test with 2048x2048 images, they were labeled in just 3 iterations.

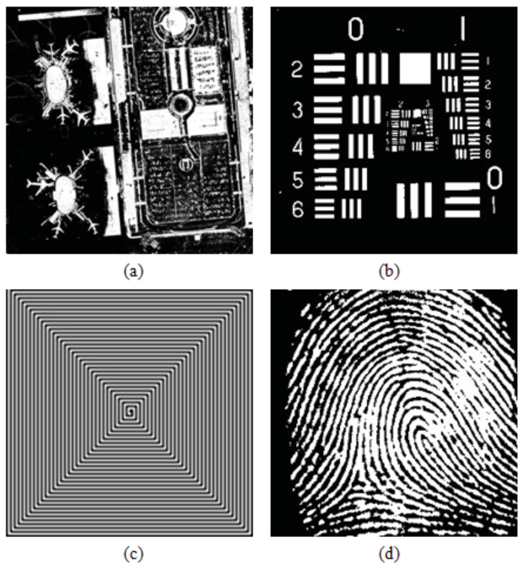


Fig. 5 Sample images in realistic image set: (a) Airport, (b) Resolution Chart from the USC image database; (c) 128x128 Spiral image; (d) Fingerprint from the FVC2000 database

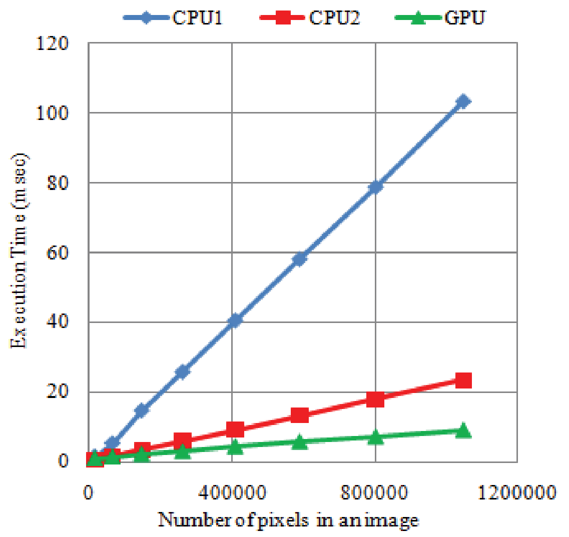


Fig. 6 Comparison of the execution times versus the size of the noise images with density 0.5.

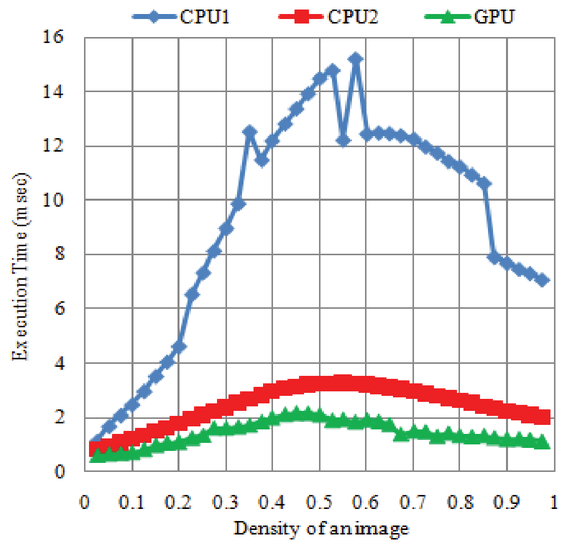


Fig. 7 Comparison of the execution times versus the density of the 384x384 noise images.

### B. *Experiment with Noise Image Set*

The experimental result from the test using the noise images with density 0.5 and various sizes is shown in Fig. 6. We can see the linearity of the execution time versus image size. In the testing the execution time versus the density of an image, 384x384 noise images with various densities were used. Proposed GPGPU algorithm (GPU) is outperforming the CPU algorithm implementations. In this case, labeling execution time of proposed algorithm is not over 2.2ms and maximum speedup is 2.1. See Fig. 7. The larger an image size is, the higher the speedup is.

### C. *Experiment with Realistic Image Set*

Table I shows various execution times on various kinds of image sizes. Our algorithm is inferior to the CPU2 algorithms for 256x256 images. The speedup of our algorithm is increasing obviously when image size is higher than 512x512.

TABLE I
COMPARISON OF VARIOUS EXECUTION TIMES (MS) ON VARIOUS KINDS OF IMAGE SIZE

| Image Size (pixel) | | CPU1 | CPU2 | Proposed |
|---|---|---|---|---|
| 256x256 | Max | 3.30 | 0.96 | 1.32 |
| | Mean | 2.05 | 0.68 | 1.01 |
| | Min | 0.69 | 0.41 | 0.54 |
| 512x512 | Max | 15.21 | 3.42 | 3.13 |
| | Mean | 8.83 | 2.43 | 2.00 |
| | Min | 2.88 | 1.33 | 1.01 |
| 512x768 | Max | 24.43 | 5.52 | 3.95 |
| | Mean | 14.72 | 3.85 | 2.73 |
| | Min | 4.50 | 2.01 | 1.41 |
| 1024x1024 | Max | 53.26 | 11.93 | 5.40 |
| | Mean | 39.35 | 9.68 | 4.19 |
| | Min | 16.39 | 6.51 | 2.97 |

## V. CONCLUSIONS

Short processing time is essential element for connected-component labeling. In this paper, proposed fast connected-component labeling algorithm is straightforward, easy to apply to the GPGPU applications and programming environment such as CUDA. In addition, our algorithm requires less memory spaces than other labeling algorithms to be suitable for implementing on GPGPU with limited memory hardware. Although performance of our algorithm with small size images is not sufficient, experiment result shows the algorithms is suitable for labeling images with massive pixels. For future work, we plan to optimize our approach to be more efficient and stable on CUDA architecture.

## ACKNOWLEDGMENT

This research was supported by the Industry Core Technology Development Project supervised by the KEIT (Korea Evaluation Institute of Industrial Technology) (KEIT-K1002153), the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency) (NIPA-2010-C1090-1001-0008), and the Seoul Research and Business Development Program, Smart (Ubiquitous) City Consortium (10561).

## REFERENCES

[1] NVIDIA Corporation, *NVIDIA CUDA$^{TM}$ Programming Guide ver. 3.0*, 2010.
[2] AMD Corporation, *Technical Overview ATI Stream Computing*, 2009.
[3] K. Suzuki, I. Horiba, and N. Sugie, “Linear-time connected-componentlabeling based on sequential local operations,”

*Computer Vision and Image Understanding*, vol. 89, pp. 1–23, 2003.

[4] K. Wu, E. Otoo, and K. Suzuki, “Optimizing two-pass connected-component labeling algorithms,” *Pattern Anal. Applic*., vol. 12, pp 117-135, 2009.

[5] L. He, Y. Chao, K. Suzuki, and K. Wu, “Fast connected-component labeling,” *Pattern Recognition*, vol. 42, pp 1977-1987, 2009.

[6] L. He, Y. Chao, and K. Suzuki, “A Linear-Time Two-Scan Labelling Algorithm,” in *IEEE International Conference on Image Processing(ICIP)*, San Antonio, Texas, USA, pp. V-241–V-244, September 2007.

[7] P. Bhattacharya, “Connected component labeling for binary images on a reconfigurable mesh architectures,” *J. Syst. Archit*., vol 42, no. 4, pp 309-313, 1996.

[8] K.A. Hawick, A. Leist, and D.P. Playne, “Parallel graph component labelling with gpus and cuda,” Massey University, Tech. Rep. CSTN-089, 2009

[9] P. Kumar, K. Palaniappan, A. Mittal, and G. Seetharaman, “Parallel Blob Extraction Using the Multi-core Cell Processor,” *ACIVS 2009, LNCS 5807*, pp 320-332, 2009.

[10] http://sipi.usc.edu/database/

[11] http://www.cs.washington.edu/research/imagedatabase/groundtruth/

[12] http://bias.csr.unibo.it/fvc2000/download.asp