International Journal of Latest Research in Science and Technology **ISSN (Online):2278-5299**
Volume 3, Issue 2: Page No.187-190 ,March-April, 2014
https://www.mnkpublication.com/journal/ijlrst/index.php

FAST PARALLEL CONNECTED COMPONENT LABELING ALGORITHMS USING CUDA BASED ON 8-DIRECTIONAL LABEL SELECTION

[1]Youngsung Soh, [2]Hadi Ashraf, [3]Yongsuk Hae, [4]Intaek Kim
[1,2,3,4]Myongji University, Yongin, Korea

Abstract- Connected component labeling (CCL) is a key step in image segmentation where foreground pixels are extracted and labeled. Sequential CCL is a computationally expensive operation and thus is often done within parallel processing framework to reduce execution time. Various parallel CCL methods have been proposed in the literature. Among them NSZ label equivalence (NSZ-LE) method seemed to perform best. In this paper we propose two new parallel CCL algorithms based on 8-directional label selection and show that they run 3 to 10 times faster than NSZ-LE depending on the characteristics of images.

Keywords – Connected Component labeling, CUDA, GPU, Parallel

I. INTRODUCTION

In computer vision, CCL is used for image segmentation to extract and label foreground pixels from background. Many approaches were proposed in the field of CCL. Wu et al. [1] divided CCL algorithms into 3 classes. They are multi-pass, two-pass, and one-pass algorithms. Multi-pass algorithm usually assumes some kind of local neighborhood to search for minimum label within that neighborhood and sometimes memorizes label equivalences. This is repeated over multiple iterations. In two-pass algorithm, 3 steps are usually executed. They are scanning, analysis, and labeling. Scanning step assigns an initial label to each pixel and records equivalence among labels if necessary while searching for neighbors. Analysis step tries to find a final label for each label by resolving equivalence chains and labeling step assigns a final label to each label. Scanning and analysis steps require one pass and labeling step requires another. One-pass algorithm scans the image from left-top to right-bottom only once and gives a new label to unlabeled pixel. Then all the pixels connected to that pixel are searched and are assigned the same label. This is repeated until no more unlabeled pixels are left. The algorithms mostly used for CCL, regardless of the number of passes they adopt, were sequential with heavy computation overhead [2, 3]. To overcome this difficulty it has been tried to implement CCL in parallel framework.

The usage of GPUs with CUDA opened a new research field for CCL and many other data processing algorithms [4]. The implementation of parallel CCL using CUDA and GPUs drastically reduced computation time. Among many parallel CCL methods using CUDA proposed so far, NSZ-LE [5] is known to perform best. However, this method requires a lot of iterations since it looks at only 4 immediate neighbors to find out the minimum label.

In this paper, we present two new algorithms for CCL using CUDA. The first method, termed as 8DLS(8-directional label selection), searches for minimum label of each object pixel for 8 directions(east, west, south, north, and 4 diagonal directions) until background pixel is encountered and changes the label of that pixel with minimum label found. This process is repeated until all object pixels have minimum labels. The second method, termed as M8DLS(modified 8DLS), is a modification of 8DLS such that object pixels having locally minimum label already will not be processed until they find that there is a change in that label, thus saving computation time.

This paper is organized as follows. In section 2, related works for CCL are discussed. Section 3 presents two proposed methods. Results are depicted in section 4 and section 5 concludes the paper.

II. RELATED WORKS

Since the early 1980s many researchers have been working on fast CCL. Some of these researches were based on sequential processing [1,2,6] and some others based on parallel computing [5,7]. Suzuki et al. [2] proposed a sequential CCL, which is quite simple and suitable for implementation in hardware. But it is not fast enough since the execution time of their method is proportional to the number of pixels in connected components in an image. So as the number of pixels in an image increases, the execution time taken by the sequential algorithm also increases, hence making it not suitable for images with high resolution.

Wu et al. [1] proposed the scan based array Union-Find (SAUF) algorithm which is basically an optimized two pass algorithm. The performance of this algorithm in accessing the memory pattern in CCL has been significantly improved by almost 10 times than that of the contour tracing algorithm [6] and other previous methods [2]. They mentioned that the only drawback of this algorithm is the immense reduction in its efficiency when images with small resolution are processed. Chang et al. [6] proposed the contour tracing algorithm which has a better computational speed than [1,2] but takes more time in accessing the memory pattern. The execution time of

Publication History

Manuscript Received	:	*25 April 2014*
Manuscript Accepted	:	*28 April 2014*
Revision Received	:	*29 April 2014*
Manuscript Published	:	*30 April 2014*

the algorithm increases with the increase in the number of connected pixels.

Hawick et al. [7] proposed parallel version of the label equivalence algorithm using GPUs. Their algorithm consists of three basic steps: scanning, analysis and labeling. These steps are repeated in a loop until all of the connected components are identified and labeled correctly. Due to the parallel implementation of CCL this algorithm decreases the execution time quite effectively but it consumes more memory in using the reference array.

Kalentov et al. [5] proposed two methods. The first one is a simple row column unification method where a single thread is assigned for each row and column, scans each row and column in a predetermined 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 have minimum labels. The second method is the NSZ-LE where a single thread is assigned to each pixel, searches for immediate 4 neighbors for minimum label, constructs the label equivalence chain, and does relabeling by resolving chains. This method seems to give the best performance among CUDA-based parallel CCL algorithms presented so far.

III.THE PROPOSED METHOD

We presented two new multi-pass methods for parallel CCL. They are 8DLS and M8DLS and are explained below in detail.

A. 8DLS Method

. This method utilizes the basic concept of 8-connectivity and checks all the 8-connected neighbors for minimum label. The pseudo code for the 8DLS method is given in Algorithm I.

ALGORITHM I: THE 8DLS METHOD IN PSEUDO

```
for j =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
         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
  if no label change for all the object pixels
  then exit
end for
```

Each pixel of the image is initialized with a unique label which is equal to the sequential index value of the pixel in the image: The iterations are the number of times the image is processed until each pixel is identified and labeled correctly. Every pixel in the input image is checked whether it is an object pixel having value ‘1’. If so, then that pixel is considered as a focused pixel. Now keeping the focused pixel in mind all 8 directions around the focused pixel are checked one by one for minimum label. For example, we check the north direction first, and then the value of the pixel connected to the focused pixel in the concerned direction will be checked. If its value is ‘0’ (meaning that it is a background pixel), then no further checking in that direction will be done. If the value of connected pixel is ‘1’, then its label will be compared with current mini of the pixel in that direction. If the label is less than current mini then that label is stored in variable mini and the label of the next pixel in that direction is checked and so on until a background pixel having value ‘0’ is hit. After that, remaining directions are checked with the same procedure.

After all 8 directions have been checked, a minimum label is obtained which is then compared with the label of the focused pixel. The minimum between the two will be assigned as the label of the focused pixel. This process is repeated for all the object pixels in the image and stops when there is no label change for all object pixels. Fig. 1 shows a running example of the 8DLS method.

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59

(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	8
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. 1 Running example of 8DLS method. (a) Initial Label, (b) After first iteration, (c) After second iteration, and (d) After third iteration.

Fig. 1(a) is a sample input image that has been uniquely 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. A single thread is assigned to each pixel and the algorithm is performed only on object pixels. Fig. 1(b), (c), and (d) show the results obtained after first, second, and third iterations respectively. To see how it works, let us consider the pixel with label 27 in Fig. 1(a). We search for 8 directions and find that the label 9(underlined) in 45° diagonal direction is the minimum. Thus the label 27 is changed to 9 as in Fig. 1(b) at the same location. For this example 3 iterations were enough to correctly label all the object pixels.

B. M8DLS Method

The basic algorithm for M8DLS is same as that of 8DLS. The pseudo Code for the M8DLS algorithm is given in Algorithm II.

ALGORITHM II: THE M8DLS METHOD IN PSEUDO

```
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 for
```

The only modification is 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, thus saving computation time. Checking for smallest is done as follows. Let initial label image array be LABEL. 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. 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 stops when there is no label change for all the object pixels in the image. The running appearance of M8DLS for initial label array in Fig. 1(a) is exactly same as the rest of Fig. 1. However, many pixels in Fig. 1(c) were not processed while producing the same results. We show Fig. 1(c) again in Fig. 2 and explain the difference. 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 a great deal of computation time.

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

Fig. 2 Running example of M8DLS method.

In M8DLS, "smallest" 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 our method.

IV. EXPERIMENTAL RESULTS

The system specification used for 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:** Images with occupancy ratio of 0.07,0.17,0.27,0.36 and 0.46 respectively
- **Spiral:** Image with spiral pattern
- **Random1 and Random2:** Images that were generated programmatically using the random number generator and have the occupancy ratio of 0.1 and 0.5 respectively.

We compared the performance of NSZ-LE [5] and two proposed methods (8DLS and M8DLS) in terms of execution time and the number iterations required. Table I shows 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. Among all the images, Spiral requires far more computation due to its complex shape.

For B1 through B5, Random1, and Random2, 8DLS performs around 3 times faster than NSZ-LE. For Spiral, 8DLS performs nearly 10 times faster than NSZ-LE. The reason for this phenomenon will be evident when we discuss the comparison results of the number of iterations required.

M8DLS shows even better performance than 8DLS. For B1 through B5, Spiral, and two random images, M8DLS shows 2%, 52%, and 17% average speed-up over 8DLS respectively.

TABLE I COMPARISON OF EXECUTION TIME (UNIT: SECOND)

Images	NSZ-LE	8DLS	M8DLS
B1	0.026	0.0081	0.00793
B2	0.028	0.0084	0.00821
B3	0.030	0.0087	0.00851
B4	0.031	0.0089	0.00874
B5	0.034	0.0093	0.00913
Spiral	0.135	0.015	0.0099
Random1	0.0297	0.0091	0.00803
Random2	0.0304	0.0098	0.00813

Table II shows the number of iterations that were taken by each method for different test images. 8DLS and M8DLS take exactly the same number of iterations, whereas NSZ-LE consumes far more iterations. This is because NSZ-LE only looks at 4 immediate neighbors, whereas both 8DLS and M8DLS consider all 8-connected neighbors. Spiral shows extreme performance difference. NSZ-LE iterates around 22 times more than the proposed methods and this causes far more computation time.

TABLE III COMPARISON OF NUMBER OF ITERATIONS REQUIRED

Images	NSZ-LE	8DLS	M8DLS
B1	6	4	4
B2	41	6	6
B3	41	6	6
B4	41	6	6
B5	41	6	6
Spiral	344	16	16
Random1	6	6	6
Random2	6	6	6

V. CONCLUSION

CCL is an important step in image segmentation and is often implemented in parallel framework to reduce execution time. Among many parallel CCL algorithms using CUDA, NSZ-LE is known to perform best [5]. We proposed two simple and fast parallel CCL algorithms using CUDA based on 8-directional label selection and showed that the proposed methods outperform NSZ-LE for various kinds of test images. The speed-up was obtained by looking at all 8-connected neighbors instead of looking at 4 immediate neighbors, thus resulting in drastic reduction in the number of iterations required with a little increase in search time. After a few number of iterations, most of pixels either has a right label already or has one in nearby location. Thus searching deep for all 8 directions may be redundant. The proposed methods may be hybridized with Kernel C algorithm [7] with some modifications and this is intended for future research.

ACKNOWLEDGMENT

This work (Grants No. C0005448) was supported by Business for Cooperative R&D between Industry, Academy, and Research Institute funded by Korea Small and Medium Business Administration in 2012.

REFERENCES

[1] K. Wu, E. Otoo, and K. Suzuki, “Optimizing two-pass connected-component labeling algorithms,” Pattern Analysis & Applications, vol. 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. 1, pp.1–23, 2003.

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

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

[5] O. Kalentev, A. Rai, S. Kemnitz, and R. Schneider, “Connected component labeling on a 2D grid using CUDA,” J. Parallel Distributed Computing, pp. 615-620, 2011.

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

[7] K. Hawick, A. Leist, and D. Playne, “Parallel graph component labeling with GPUs and CUDA,” Parallel Computing, vol. 12, pp. 655-678, 2010.