A Parallel Algorithm for Connected Component Labelling of Gray-scale Images on Homogeneous Multicore Architectures

2010 J. Phys.: Conf. Ser. 256 012010

(http://iopscience.iop.org/1742-6596/256/1/012010)

Journal of Physics: Conference Series **256** (2010) 012010 doi:10.1088/1742-6596/256/1/012010

A Parallel Algorithm for Connected Component Labelling of Gray-scale Images on Homogeneous Multicore Architectures

Mehdi Niknam[1,2], Parimala Thulasiraman[1,2], Sergio Camorlinga[2,1]

[1]Department of Computer Science - University of Manitoba, [2]Telecommunication Research Labs

mhniknam@cs.umanitoba.ca, thulasir@cs.umanitoba.ca, scamorlinga@trlabs.ca

Abstract. Connected component labelling is an essential step in image processing. We provide a parallel version of Suzuki's sequential connected component algorithm in order to speed up the labelling process. Also, we modify the algorithm to enable labelling gray-scale images. Due to the data dependencies in the algorithm we used a method similar to pipeline to exploit parallelism. The parallel algorithm method achieved a speedup of 2.5 for image size of 256 X 256 pixels using 4 processing threads.

1. Introduction

Computer-aided diagnosis (CAD) systems are examples of diagnostics, screening and detection tools for medical purposes. CAD provides clinicians (e.g. radiologists) a computerized analysis of medical images as a 'second opinion' in detecting lesions, assessing extent and progression of disease, and supporting diagnostic decisions among other things. CAD systems utilize image processing techniques in order to detect and diagnose different diseases from medical images.

The pre-processing techniques such as image filtering and image registration play an important role in enhancing the accuracy of image analysis and later steps in CAD systems. Connected component labeling is a useful tool used in pre-processing stages as well as in image analysis and in post processing stages [1]. In connected component labeling of an image, every set of connected pixels having same gray-scale values are assigned the same unique region label. This region later will be used to identify the suspicious lesions. The fact that the connected component labeling is a fundamental module in medical image processing, optimizing the existing algorithm will result in an improvement of many medical diagnoses and procedures [2, 3, 4].

Medical applications are computationally and data intensive problems. With the recent advancement in multicore architectures, these problems are gaining insight from a whole new perspective [5]. The focus of this paper is to parallelize the sequential connected component labeling algorithm presented by Suzuki et al. [6] on a homogenous multicore architecture in order to study the speedup of the parallel algorithm.

We use OpenMP to parallelize the algorithm. We also apply a similar method mentioned in [1] to enhance Suzuki's algorithm to label the gray-scale images since the original algorithm labels only binary images and the medical images are normally gray-scale. Section 2 provides a brief discussion regarding the existing connected component labeling algorithms. Section 3 presents Suzuki's algorithm in detail. Our parallel algorithm is explained in the section 4. Section 5 outlines the experimental framework we used and the section 6 provide a discussion of the experimental results. Finally, section 7 describes conclusion and future work.

2. Background and Related Work

Suzuki et al. [6] classified the connected component labeling methods into 4 different categories: A) Algorithms that repeat passes through an image in forward and backward directions alternately to propagate the label equivalences until no labels change. B) Algorithms using two passes in a way that they assign provisional labels to the connected components and store the label equivalences in an array. Then they resolve the label equivalences using a search algorithm such as union-find algorithm [7] and store the resolved result in a one-dimensional table. In the second pass, they replace the provisional labels with the smallest equivalent label. C) Algorithms that represent the image using hierarchical tree structures and resolve the label equivalences using union-find algorithm. D) And parallel algorithms developed for specific parallel architectures such as mesh and hypercube parallel processors.

Suzuki et al. [6] introduce an algorithm which optimizes the algorithms in category A. Suzuki's algorithm promises a linear time complexity. Suzuki et al. show that maximum of four scans is sufficient for the images with complex geometrical shapes. Wu et al. [8, 9] provide an optimization on Suzuki algorithm that reduces the number of neighboring pixels needed to be examined during the scans. He et al. [10], present a similar algorithm which scans the whole image once and then resolves the label equivalences using recorded run data.

We chose the Suzuki algorithm to parallelize due to its linear time complexity. There are several sequential algorithms to enhance the performance of the Suzuki algorithm; however, due to our best knowledge there is no parallel version for this algorithm. We decided to investigate the performance improvement of the parallel version of the algorithm.

3. Suzuki Algorithm

The Suzuki algorithm scans through the image in forward and backward directions using masks shown in Figure 1, in order to assign a provisional label to each pixel. Also, it stores the label equivalences in an additional one dimensional array called label connection table. The provisional labels propagate through the image as well as the label connection table that reduces the number of scans needed to complete the labeling.

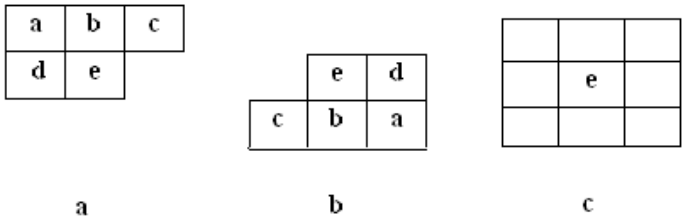


Figure 1- a) Forward scan mask b) backward scan mask c) 8-connected neighborhood

Suppose that a binary image *b(x,y)* consists of pixel values F_O(indicating objects), and F_B (indicating the background); and a provisional label *m* is initialized to 1. In the first scan, the Suzuki algorithm assigns a provisional label to each pixel at position "e" according to the following equation:

$$g(x,y)=\begin{cases} F_B & \text{if } b(x,y)=F_B, \\ m,(m=m+1) & \text{if } \forall\{i,j\in M_s\}g(x-i,y-j)=F_B, \\ T_{\min}(x,y) & \text{Otherwise} \end{cases}$$

$$T_{\min}(x,y)=\min\left[\{T[g(x-i,y-j)]\,|\,i,j\in M_S\}\right]$$

Where *g(x,y)* stores the provisional labels, *T[m]* is the label connection table, (*m=m+1*) indicates an increment of *m*, min(.) an operator calculating the minimum value, and M_S the region of the mask except the object pixel, i.e. *b(x-1,y-1), b(x,y-1), b(x+1,y-1), and b(x-1,y).*

Also, the label connection table *T[m]* is updated at the same time as *g(x,y)* according to the following equation:

$$\begin{cases} \text{non-operation} & \text{if } b(x,y) = F_B, \\ T[m] = m & \text{if } \forall\{i,j \in M_s\} g(x-i,y-j) = F_B, \\ T[g(x-i,y-j)] = T_{\min}(x,y) & \text{if } g(x-i,y-i) \neq F_B. \end{cases}$$

In the next scans, the forward and backward scans are performed alternately and the tables are updated using the following equations:

$$g(x,y) = \begin{cases} F_B & \text{if } g(x,y) = F_B, \\ T_{\min}(x,y) & \text{otherwise,} \end{cases}$$

$$T_{\min}(x,y) = \min[\{T[g(x-i,y-j)] \,|\, i,j \in M\}].$$

$$\begin{cases} \text{non-operation} & \text{if } g(x,y) = F_B, \\ T[g(x-i,y-j)] = T_{\min}(x,y) & \text{if } g(x-i,y-j) \neq F_B. \end{cases}$$

The forward and backward scans will be performed repeatedly while the following condition is not satisfied:

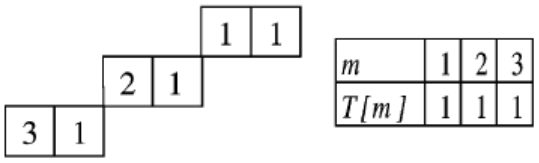
$$g(x-i,y-j) \neq T_{\min}(x,y) \quad \text{if } g(x-i,y-j) \neq F_B,$$

$$i,j \in M_S.$$

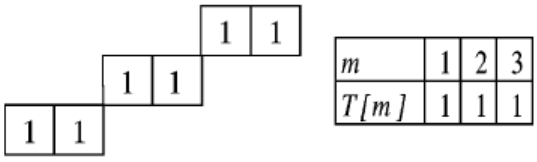
The end result is the image with the final label for each pixel. Figure 2 shows an example of labeling according to the Suzuki algorithm.

0	0	0	0	1	1
0	0	1	1	0	0
1	1	0	0	0	0

a) A binary image to be scanned



b) After first scan



c) After second scan

Figure 2- Example of labeling using Suzuki algorithm

The following is a pseudo code of the Suzuki algorithm:

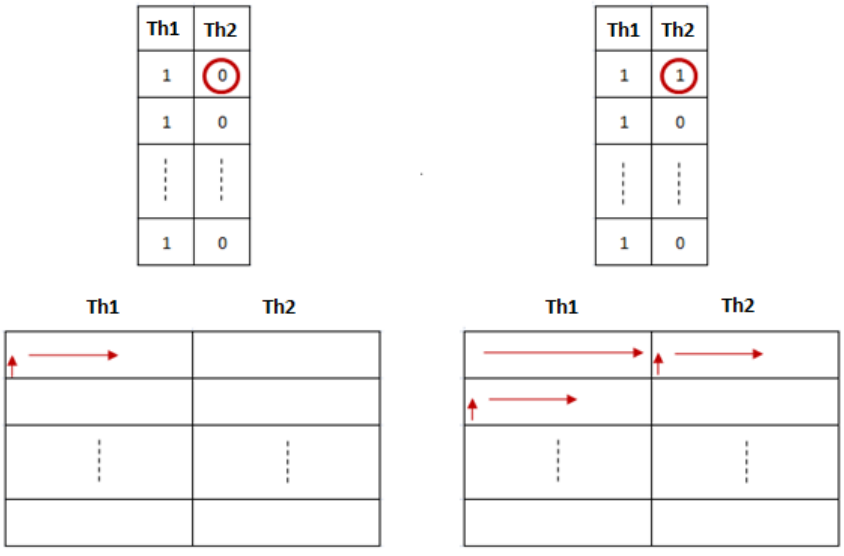
```
algorithm Suzuki(data)
   First pass
   for row in data
       for column in row
           assign a label to data[row][col] using Forward Mask
   Next passes
   while there is a change in labels do
      for row in data
         for column in row
             update the labels[row][column] using Backward Mask
      for row in data
         for column in row
             update the labels[row][column] using Forward Mask
return labels
```

4. Solution Strategy

In this section we describe our proposed method to parallelize the Suzuki algorithm and its implementation in OpenMP.

4.1. Proposed Method

The Suzuki algorithm is sequential in nature. That is, the result of the previous iteration will be used for the next iteration. Two data structures storing the provisional label and the label connection table (see section 3) are updated after labeling a pixel and their updated versions are used to label the next pixel. Due to these data dependencies, it is not possible to perform the labeling using multiple threads without specifying the order of execution. We used a method similar to pipeline in order to exploit parallelism. The following is a detail description of the proposed method to parallelize the algorithm.



a) Th1 labeling its portion and Th2 is waiting b) Th1 and Th2 are labeling concurrently

Figure 3- Example of the algorithm with 2 threads

As indicated in the pseudo code, the algorithm examines each pixel's neighbors and assigns a provisional label to it accordingly. In our algorithm, the image pixels are distributed between threads row-wise. Assuming there are 2 threads (Th1 and Th2) and the image has 128 rows and 128 columns. The first 64 pixels in the first row will be labeled by Th1 and the next 64 pixels by Th2. However, Th2 cannot start labeling in the same row as Th1, until Th1 is done with labeling its portion due to the sequential nature of the Suzuki algorithm. The Th2 can start labeling when Th1 is done, also Th1 can start labeling the first 64 pixels of the second row at the same time. After Th1 is done with labeling the first portion of the second row, Th2 can label its portion of the second row, and so forth. With this method, we maintain the order of execution of the algorithm as well as exploiting parallelism.

In order to synchronize the thread in the manner mentioned above, we utilized a two dimensional array of size: "number of thread" by "number of rows" called "condition". The thread cannot label its portion of a row and need to wait, if the corresponding value in the array is 0. As soon as Th1 finished the labeling of its portion, it will change the value for Th2 in the condition array to 1. Then Th2 labels its portion and Th1 labels its portion in the next row. Figure 3 shows the image and the condition array for two iterations.

4.2. OpenMP Implementation

As can be seen in the iterations example in section 4.1, after the first iteration Th1 and Th2 label their portions concurrently. However, there is some overhead for the synchronization. Each scan in the sequential algorithm includes two loops. In order to parallelize the scan portion of the algorithm in OpenMP as described in the example, we can add *pragma omp parallel for* on the inside loop as shown below:

```
for row in data
    #pragma omp parallel for
    for column in row
        while the condition is not 1 for this thread wait;
        label the pixel;
        if it is end of the row set the condition of the next thread to 1;
```

In the above code, the data will be distributed properly; however the overhead is very high. The number of forking and joining in this code is equal to the number of columns in the image which causes high overhead. In order to deal with this overhead, we use *pragma omp parallel* on the top of the first loop. The cost for the *pragma omp parallel* is only one time fork and joins. We also provide a suitable code to perform the data distribution in the program. Following is the parallel pseudo code that is implemented for the scan portions of the algorithm:

```
chunk = row / number of threads;
#pragma omp parallel
for row in data
    k = calculate the start index for this thread;
    while the condition is not 1 for this thread wait;
    for k in row to k + chunk
        label the pixel;
    set the condition of the next thread to 1;
```

In addition, we used *pragma omp critical* in order to synchronize the access to the label connection table (T[m]). The *pragma omp critical* provides synchronization in a way that only one thread can update the T[m] at a time. There are three scan loops in the Suzuki algorithm as shown in section 3. All three scan loops are replaced with above parallel code. We made some modifications to the Suzuki algorithm to enable labeling of gray-scale images. To distinguish between region of interest (ROI) and

background in the images we defined a threshold value. We also modified the algorithm to compare gray values instead of binary values.

5. Experiment

We performed a set of experiments to measure the speedup of our proposed parallel version of Suzuki's algorithm. We used 30 gray-scale images in 3 sizes: 128 X 128, 256 X 256, and 512 X 512 pixels from University of Massachusetts Vision Image Archive [11]. The images were labeled using the sequential and parallel algorithm. We examine the sequential algorithm against parallel version using four different numbers of threads: 2, 4,8,16.

The algorithms were implemented in C and OpenMP. The algorithms were tested on one of the five computing nodes of a cluster called helium. The helium cluster has one head node (Sun Fire X4200) and five computing nodes. Each computing node is a Sun Fire X4600 machine which has eight Dual-Core (16 Cores) AMD Opteron 885 2.6 GHZ.

6. Results

The Figure 4 shows the speedup of the parallel version of the algorithm. For the small size images (128 X 128), the sequential version is faster than the parallel version. The reason of no speedup for small size images is due to the overhead caused by synchronization of the threads.

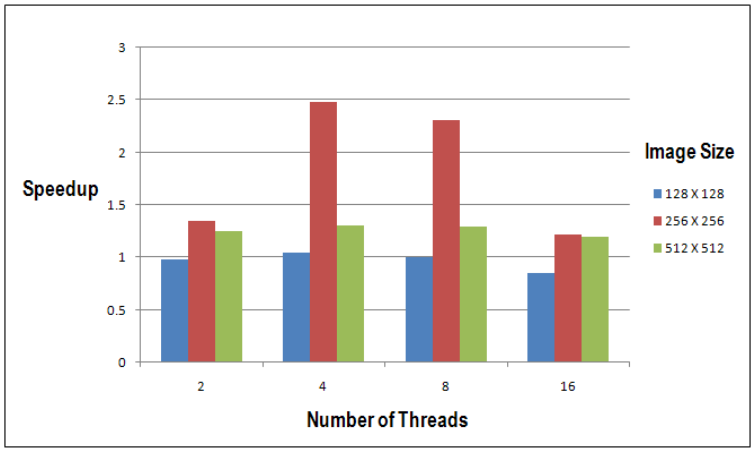


Figure 4- The speedup for different image sizes and different numbers of threads

For the medium size images (256 X 256), the algorithm achieved the best speedup in comparison to other image sizes. The best speedup of 2.5 belongs to the 4 threads while labeling the medium size images. However, in medium size images using 2 and 16 threads the speed up is not significant. While using 16 threads the size of data allocated to each thread reduces. Consequently, the overhead is higher than the amount of exploited parallelism.

For large size images (512 X 512), the speedup is almost similar in all different number of threads. As can be seen there is a trade-off between the data size and the number of threads. The synchronization cost will not be covered by parallelism exploitation in small size images. Similarly, the synchronization overhead is high when using too many threads since the data size allocated to each thread reduces.

According to the results, the overhead of our approach is not decreased linearly with the problem size. One possible reason could be an additional overhead caused by cache miss for image size of 512 X 512 since the image cannot be transferred completely to the cache. The other possible reason could be an extra overhead caused by Non-Uniform Memory Access (NUMA) effect related to the specific architecture used to test the algorithm.

Considering the sequential nature of the Suzuki algorithm, the speedup achieved for medium size (256 X 256) is reasonable and it can be used to label specific medical images. However, to achieve a higher performance as a future work we can investigate using a method which divides the image to number of processes and run the Suzuki algorithm locally, then provide a way to merge the portions in order to achieve a consistent labeling for the image.

7. Conclusion

Connected component labeling is a fundamental module in image processing. We provide a parallel version of Suzuki's sequential connected component algorithm in order to speed up the labeling process. We also modified the algorithm to enable labeling of gray-scale images. Due to the data dependencies in the algorithm, we used a method similar to pipeline in order to exploit parallelism. The parallel algorithm achieved the best speedup when labeling medium sized images. Explicitly, the speedup of 2.5 is achieved while labeling image size of 256 X 256 using 4 threads.

As a future work, we can investigate a method to provide a parallel connected component algorithm which divides the image between several threads and each thread utilizes the Suzuki algorithm enhancement ideas proposed in this paper for its local data. Then we integrate the local results to provide a consistent label for the whole image. Also, examining a connected component algorithm on other parallel architectures such as Graphics Processing Unit (GPU) can be performed.

References

[1] R. D. Yapa, K. Harada, " Connected Component Labeling Algorithms for Gray-Scale Images and Evaluation of Performance using Digital Mammograms*," IJCSNS International Journal of Computer Science and Network Security,* **8**(6), pp. 33–41, 2008.

[2] J. Freixenet, X. Mu˜noz, D. Raba, M. Marti, and X. Cuf´ı, "Yet another survey on image segmentation: Region and boundary information integration," in *Proceedings of the European Conference on Computer Vision (ECCV 2002)*, pp. 408–422, 2002.

[3] B. van Ginneken, B. M. ter Haar Romeny, and M. A. Viegever, "Computer-aided diagnosis in chest radiography: A survey," *IEEE Transcations on Medical Imaging* **20**(12), pp. 1228–1241, 2001.

[4] T. W. Nattkemper, "Automatic segmentation of digital micrographs: a survey," in *Proceedings of MEDINFO 2004, San Francisco*, *Americal Medical Informatics Association*, 2004.

[5] M. Knaup, S. Steckmann, O. Bockenbac and M. Kachelriess, " Tomographic Image Reconstruction using the Cell Broadband Engine (CBE) General Purpose Hardware", *Proceedings Electronic Imaging, Computational Imaging V*, *SPIE* Vol 6498, 64980P, pp. 1-10, January 2007.

[6] K. Suzuki, I. Horiba, and N. Sugie, "Linear-time connected-component labeling based on sequential local operations," *Comput. Vis. Image Underst.* **89**(1), pp. 1–23, 2003.

[7] C. Fiorio, J. Gustedt, "Two linear time union-find sterategies for image processing," Theoretical Computer Science, **154**(2), pp. 165–181, Feb. 1996.

[8] K. Wu, E. Otoo, and A. Shoshani, Optimizing Connected Component Labeling Algorithms, In Proceedings of SPIE Medical Imaging Conference, pp. 1965-1976, Apr. 2005.

[9] K. Wu, E. Otoo, and K Suzuki.: Optimizing two-pass connected-component labelling algorithms. *Pattern Analysis and Applications* 12, pp.117-135, 2009

[10] L. He, Y. Chao, K. Suzuki, T. Nakamura, and H. Itoh: A label-equivalence-based one-scan labeling algorithm. Journal of Information Processing Society of Japan 50, pp. 1660-1667, 2009.

[11] Computer Vision Research Laboratory at UMass. University of Massachusetts Vision Image Archive, http://vis-www.cs.umass.edu/~vislib/, Retrieved May 2010.

[12] OpenMP. The OpenMP API Speccification for Parallel Programming, http://openmp.org/wp, Retrieved May 2010.

[13] OpenMP. OpenMP tutorial, https://computing.llnl.gov/tutorials/openMP/, Retrieved May 2010.