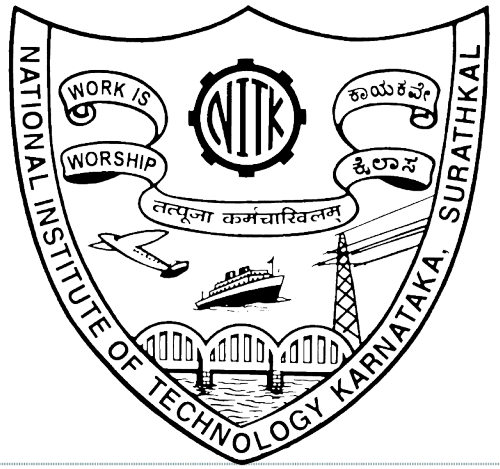


National Institute of Technology Karnataka

Parallel Computing Project Presentation

CONNECTED COMPONENT LABELLING

Under the guidance of

Dr Geetha V.
Assistant Professor
Department of Information Technology

Presented By:

Divija Nagaraju
14IT112
Mukta Kulkarni
14IT220
Pooja Soundalgekar
14IT230

Contents

- Introduction
- Problem Statement
- Objective
- Methodology Used
- Implementation
- Conventional and Suzuki's Algorithm
- Two pass algorithm
- Result
- Future Work
- Conclusion
- References

Introduction

- Connected-component labelling (alternatively called region extraction)
- Algorithmic application of graph theory.
- Subsets of connected components are uniquely labelled based on a given heuristic.
- It is used in computer vision to detect connected regions in binary digital images.
- Colour images and data with higher dimensionality can also be processed.

Introduction (contd.)

- There exist many algorithms for computing connected components in a given image.
- Repeated pass algorithms
- Two-pass algorithms
- Algorithms with the hierarchical tree equivalent representations of the data parallel algorithms.

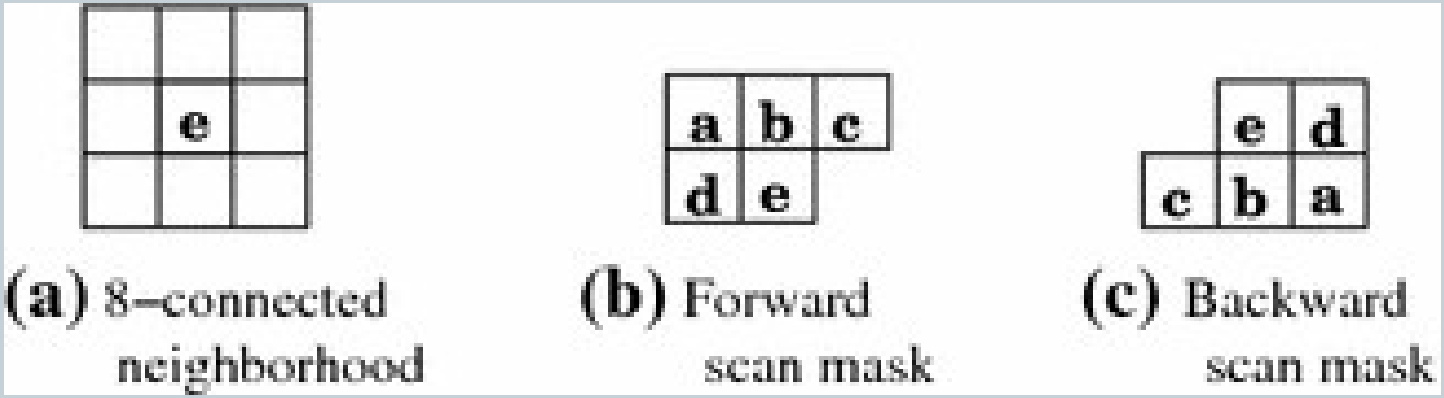
Problem Statement

- The project aims to compute the connected components in a binary image and label them.
- The implementation of the algorithm has been done using thread level parallelism and pipelined architecture.

Objective

- The computational speed for labelling the binary image is compared to the serial implementation computational speed.
- Based on the result conclusive remarks are analysed and future work is being proposed.

Methodology Used



(a) 8–connected neighborhood (b) Forward scan mask (c) Backward scan mask

CONVENTIONAL AND SUZUKI'S ALGORITHM

Serial Implementation

The conceptual algorithm is given as follows :

```
1. Algorithm (data)
2.       First pass
3.           for row in data
4.               for column in row
5.                   assign a label to data[row][col] using Forward Mask
6.     Next passes
7.         while there is a change in labels do
8.             for row in data
9.               for column in row
10.                  update the labels[row][column] using Backward Mask
11.          for row in data
12.           for column in row
13.                  update the labels[row][column] using Forward Mask
14. return labels
```

CONVENTIONAL AND SUZUKI'S ALGORITHM

Parallel Implementation

```
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;
```

TWO PASS ALGORITHM
Serial Implementation

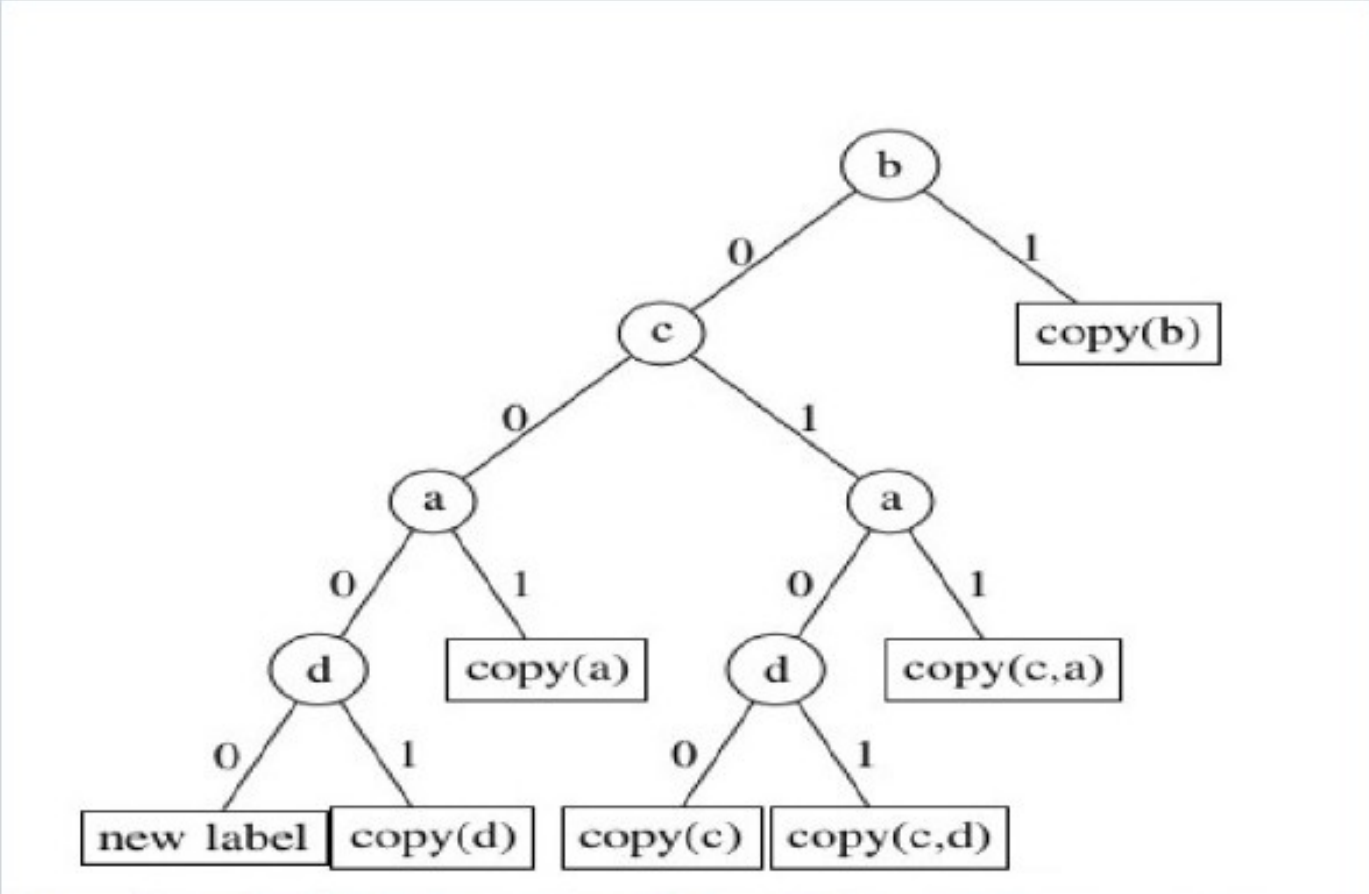
Main Function

Input: $2D$ array $image$ containing the pixel values
Output: $2D$ array $label$ containing the final labels

```
1: function CCLREMSP(image)
2:     Scan_CCLRemSP(image) ▷ Scan Phase of CCLREMSP
3:     ▷ count is the max label assigned during Scan Phase
4:     flatten(p, count)          ▷ Analysis Phase of CCLREMSP
5:     for row in image do        ▷ Labeling Phase of CCLREMSP
6:         for col in row do  ▷ e is the current pixel to be labeled
7:             label(e) ← p[label(e)]
8: end function
```

TWO PASS ALGORITHM
Serial Implementation(Contd)

Decision tree used



TWO PASS ALGORITHM
Serial Implementation (Contd)

Scan Function

Input: $2D$ array *image* containing the pixel values
InOut: $2D$ array *label* containing the provisional labels and $1D$ array p containing the equivalence info
Output: maximum value of provisional label in *count*

```
 1: function SCAN_CCLREMSP(image)
 2:     for row in image do
 3:         for col in row do
 4:             if image(e) = 1 then
 5:                 if image(b) = 1 then
 6:                     copy(b)
 7:                 else
 8:                     if image(c) = 1 then
 9:                         if image(a) = 1 then
10:                             copy(c, a)
11:                         else
12:                             if image(d) = 1 then
13:                                 copy(c, d)
14:                             else
15:                                 copy(c)
16:                     else
17:                         if image(a) = 1 then
18:                             copy(a)
19:                         else
20:                             if image(d) = 1 then
21:                                 copy(d)
22:                             else
23:                                 new label
24:     return count
25: end function
```

TWO PASS ALGORITHM

Parallel Implementation

Input: $2D$ array $image$ containing the pixel values
Output: $2D$ array $label$ containing the final labels

```
1:  function PAREMSP(image)
2:      numiter ← row/2 ▷ As we are processing 2 rows at a time
3:      # pragma omp parallel
4:      chunk ← numiter/numberofthreads
5:      size ← 2 × chunk
6:      start ← start index of the thread
7:      count ← start × col
8:      # pragma omp for
9:      Scan_ARemSP(image)
10:     # pragma omp for
11:     for i = size to row − 1 do
12:         for col in row do
13:             if label(e) ≠ 0 then
14:                 if label(b) ≠ 0 then
15:                     merger(p, label(e), label(b))
16:                 else
17:                     if label(a) ≠ 0 then
18:                         merger(p, label(e), label(a))
19:                     if label(c) ≠ 0 then
20:                         merger(p, label(e), label(c))
21:         i ← i + size
22:     flatten(p, count)
23:     for row in image do
24:         for col in row do
25:             label(e) ← p[label(e)]
26: end function
```

Use of Critical Section

- In Suzuki algorithm : Assign label in forward and backward mask must be handled using critical section.
- In Two Pass Algorithm : Flatten function must be handled using critical section directive

Result

INPUT ▼	SERIAL		PARALLEL	
	SUZUKI	TWO PASS	SUZUKI	TWO PASS
1.	0.9122 ms	0.8841 ms	0.3575 ms	0.1902 ms
2.	0.9932 ms	0.8461 ms	0.5092 ms	0.5563 ms
3.	0.5832 ms	0.2000 ms	0.0681 ms	0.0332 ms

INPUT 1

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

INPUT 2

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

INPUT 3

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

Future Work

- A research of other architectures apart from the pipeline architecture used in this algorithm that may reduce communication costs.
- Better understanding of the performance features and trade-offs of these strategies.
- Apply other parallelisation strategies such as divide and conquer and tree architectures to optimize the parallel architecture.

Conclusion

- The task of connected components labelling is an important step in multiple image processing and computer vision projects.
- Reducing the speed of computation of this task while maintaining it's accuracy is indeed a necessity.
- Analyses show that a two-pass algorithm using the strategies implemented has the optimal worst-case time complexity O(p)
- Thus it can be implemented in practical scenarios.

Individual Contribution

- Divija : Suzuki Serial and Two Pass Serial
- Mukta : Two Pass Algorithm Serial and Two Pass Parallel
- Pooja : Suzuki Algorithm Serial and Parallel

References

[1]. 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.

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

[3].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.

[4].Siddharth Gupta, Diana Palsetia, ”A New Parallel Algorithm for 2 pass labelling of connected components”-ArXiv 20 June 2016

[5].Mehdi Niknam and Sergio Camorlinga, “A Parallel Algorithm for Connected Component Labelling of Grayscale Images on Homogeneous Multicore Architectures ”, High Performance Computing Symposium-2010