Connected Component Labeling Algorithm for very complex and high resolution images on an FPGA platform

Kurt Schwenk* and Felix Huber

German Aerospace Center (DLR), German Space Operations Center (GSOC), Münchner Str. 20, 82234 Wessling, Germany

ABSTRACT

Connected Component Labeling (CCL) is a basic algorithm in image processing and an essential step in nearly every application dealing with object detection. It groups together pixels belonging to the same connected component (e.g. object). Special architectures such as ASICs, FPGAs and GPUs were utilised for achieving high data throughput, primarily for video processing. In this article, the FPGA implementation of a CCL method is presented, which was specially designed to process high resolution images with complex structure at high speed, generating a label mask. In general, CCL is a dynamic task and therefore not well suited for parallelisation, which is needed to achieve high processing speed with an FPGA. Facing this issue, most of the FPGA CCL implementations are restricted to low or medium resolution images ($\leq 2048 * 2048$ pixels) with lower complexity, where the fastest implementations do not create a label mask. Instead, they extract object features like size and position directly, which can be realized with high performance and perfectly suits the need for many video applications. Since these restrictions are incompatible with the requirements to label high resolution images with highly complex structures and the need for generating a label mask, a new approach was required. The CCL method presented in this work is based on a two-pass CCL algorithm, which was modified with respect to low memory consumption and suitability for an FPGA implementation. Nevertheless, since not all parts of CCL can be parallelised, a stop-and-go high-performance pipeline processing CCL module was designed. The algorithm, the performance and the hardware requirements of a prototype implementation are presented. Furthermore, a clock-accurate runtime analysis is shown, which illustrates the dependency between processing speed and image complexity in detail. Finally, the performance of the FPGA implementation is compared with that of a software implementation on modern embedded platforms.

Keywords: CCL, FPGA, on-board image processing

1. INTRODUCTION

1.1 Motivation

The labeling of the connected components of an image is a fundamental processing step in object recognition. Pixels which belong to the same connected component are grouped together and indexed with a unique label, as can be seen in figure 1. For a feasibility study of a future on-board analysis system for optical satellite data, based on a field-programmable gate array (FPGA), a CCL-method capable of a processing high resolution images with very complex image structures, should be implemented. Since image processing tasks are often computationally intensive, the FPGA has been chosen as system architecture. FPGAs can deliver enough computational power at a relative small power consumption. Butqqq:q:q, therefore, the processing tasks have to be well suited to the FPGA environment. The processing tasks should have the potential for parallelisation and they should be implementable with mostly static local operations. For example, basic window filters like gaussian- or sobel-filters, are very suitable for an FPGA implementation, but it is difficult and time-consuming to implement more complex, dynamic and global processing tasks like Connected Component Labeling (CCL) with good performance and moderate resource consumption on an FPGA. Therefore, sophisticated research has to be performed to find an appropriate labeling method and to implement this method in a beneficial way.

*Send correspondence to Kurt Schwenk
E-mail: Kurt.Schwenk@dlr.de, Telephone: (+49) 0 8153 28 3493

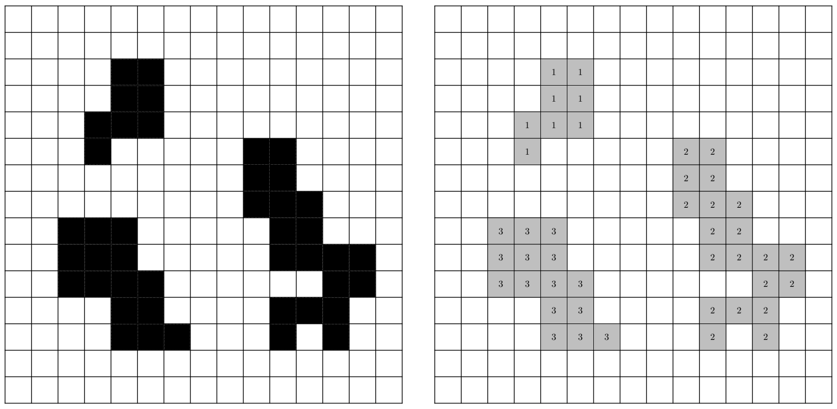


Figure 1. left binary image, right label mask

1.2 Content

In this paper, the implementation of a CCL-method on an FPGA is presented, which was designed for the multimodal labeling of high resolution images with no restrictions regarding image content. These requirements differ from that of nearly all FPGA CCL-implementations, which where primarily designed for video processing. For video application, typically binary images with a resolution in most cases much lower than 2048×2048 pixels, have to be labeled with more than 30 frames per seconds (FPS). The processing time and memory usage of most of the common CCL-methods is noticeably dependent on the image content. To reach a stable high FPS count and to lower resource consumption, CCL-implementations for video processing on FPGAs are restricted either in the number, the size, or the form of the connected components in the image. These limitations have no influence on standard video applications. Furthermore, the highest-performance FPGA CCL-implementations, are directly extracting information of the connected components, such as for example bounding boxes, the volume or the center of gravity. This can be done much more efficiently than creating a full label mask. For many image analysis tasks, a label mask is needed and restrictions to image content are undesirable. Therefore, a CCL-method has to be implemented, which is capable of labeling very big images with no restriction on image content in a reasonable time. In this paper, the basics of CCL are explained. Afterwards, an overview of common labeling methods including a suitability study is presented. Finally the implemented method is presented with a detailed resource requirement and performance analysis.

2. BASICS

The input for connected component labeling is local connectivity information. For every pixel this is the information about which neighbored pixels are connected. In the case of a binary image, two neighboring pixels are called connected, if they are both white or black. Background pixels are often ignored, as shown in figure 1. In the case of multimodal labeling, two neighboring pixels have the freedom to be connected or not, as illustrated in figure 2. Note that the label mask shown in this figure can not be the result of a binary CCL-operation. Two arbitrary pixels are connected if there is a path between neighboring connected pixels, as illustrated in figure 3. A connected component is a set of all pixels in an image, which are connected with respect to the defined connectivity. There are two common types of connectivity. In the case of 4-connectivity, only the pixels directly above, below, right and left are counted as neighbors, whereas in the case of 8-connectivity the four adjacent diagonal pixels are counted as neighbors, also. There can be a slight difference in the labeling result depending on using 4- or 8-connectivity, as shown in figure 4.

3. CCL-METHODS

In the following, common labeling methods are presented. A general overview of modern methods can be found for example in Lacassagne and Zavidovique.[1] For a run-time analysis, a set of real and synthetic test images was used. For the synthetic test, tiled images were used. The tiles used are shown in the Appendix, figure 21 and

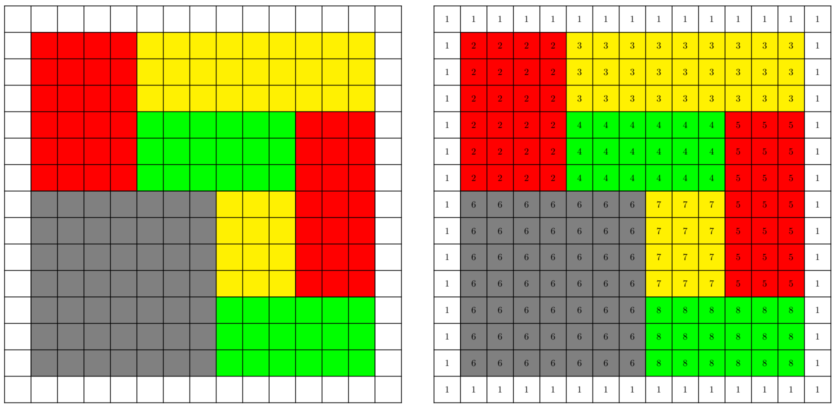


Figure 2. multimodal connected component labeling

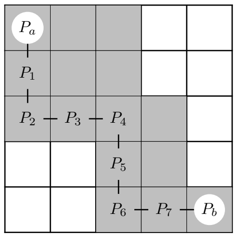
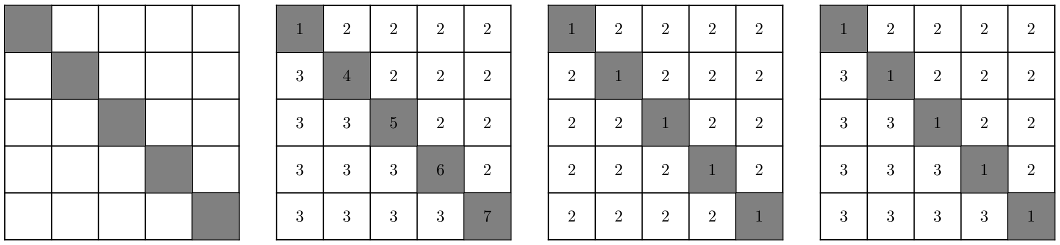


Figure 3. connectivity between two points



(a) Orginal (b) 4-connectivity (c) 8-connectivity (d) neither 4- nor 8-connectivity

Figure 4. Example of 4-, 8-connectivity

an example test image in figure 22. The real test set consists of 365 binary Landsat TM/ETM+ ACCA cloud masks with sizes around 8000×8000 pixel.

3.1 Multipass methods

Multipass labeling is a fully iterative window based method. It was first presented in 1966 by Rosenfeld and Pfaltz,[2] an advancement 1981 by Haralick.[3] As illustrated in figure 5, the image is first scanned in forward direction with a forward mask, and after that in backward direction with a backward mask. During the scanning pass, the label of the central point of the mask is set to the smallest label of the points of the mask connected with the center point. During the first pass, a new label is created if no point of the mask is connected with the center point. Forward and backward scanning is repeated until no changes occur anymore. Then the labeling is complete. An example is shown in figure 6. In this example two passes are needed for completing the connected component labeling. An FPGA implementation of this method for binary image labeling was published 1999 by Crookes and Benkrid.[4] Since multipass labeling is purely windows based, it is very well suited for an FPGA implementation. But a thorough analysis with the synthetic and real test data revealed, that the multipass method is not suitable for the labeling of large images with complex structures. As can be seen in figure 7, for

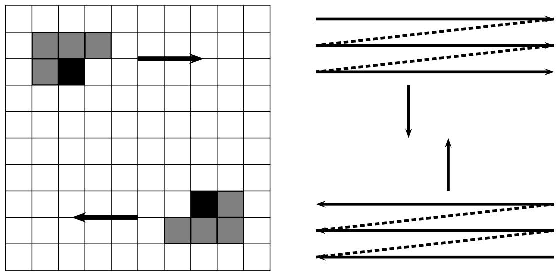


Figure 5. left image: top left forward-mask, bottom right backward-mask, black central point; right image: top forward-scan, bottom backward-scan

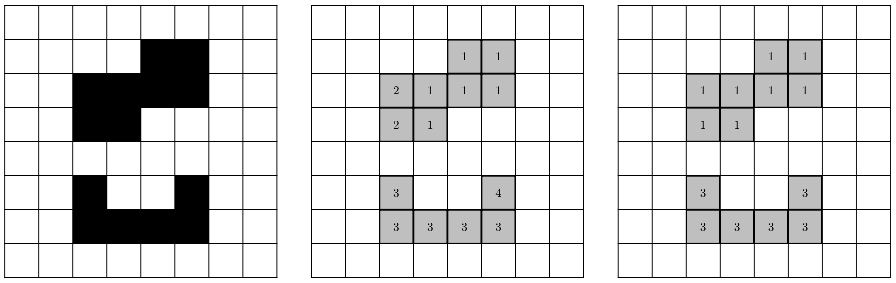


Figure 6. Example multipass labeling, 8-connectivity, left: binary image, center: label mask after forward-scan, right: label after backward-scan

images with complex structures, a high number of passes are necessary. In assumption that an FPGA solution can process $100 * 10^6$ pixels per second[*], more than 30 minutes for the labyrinth-2 test image and over 90 minutes for the diagonal-2 test image with a size of $8192 * 8192$ pixels are required. For the real test images up to 300 passes were required.

Syn. test images	Number of passes						
Width/Height in pixel	50	512	1024	2048	4096	8192	16384
White	2	2	2	2	2	2	2
Diagonal 1	4	4	4	4	4	4	4
Diagonal 2	2	2	2	2	2	2	2
Diagonal 2 (rotated 90° left)	47	509	1021	2045	4093	8189	-
Triangle	2	2	2	2	2	2	2
Labyrinth 1	23	23	23	25	23	23	23
Labyrinth 2	23	186	363	745	1479	2943	5889

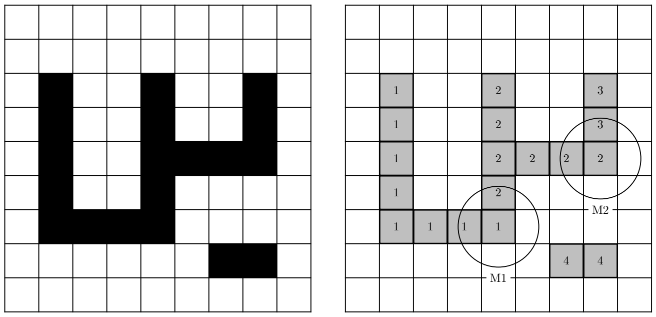
Figure 7. Multipass method, synthetic test images, number of passes

3.2 Two pass methods

Two pass connected component labeling is a classical method, mentioned already 1966 in a publication from Rosenfeld and Pfaltz.[2] An extensive description can be found in reference books such as Computer and robot vision from Haralick and Shapiro.[5] Classical implementations of two pass methods need two scanning passes and an intermediate step for conflict resolution. An example is shown in figure 8. The forward scan is equal to the forward scan of the multipass labeling. The center point of the mask gets the lowest label of the pixels in the mask connected with it. If no pixel is connected with the center point, a new label is assigned. In figure 8, the result of the first scanning pass is depicted in the top right image. A conflict occurs if two or more pixels in the scanning mask that are connected with the center point, have different labels. This conflict is recorded in

[*]equals an FPGA running at 100 MHz processing one pixel per clock

an equivalence table. After the first pass, a look up table (LUT) is created assigning final labels to temporary labels. This process is referred to as equivalence resolving. In the second pass, the labels of the label mask are updated using the LUT. The critical point of two pass methods is the resolving of the equivalences. With a lot



Merge Table	
M1	(1,2)
M2	(2,3)

Look Up Table	
1	1
2	1
3	1
4	4

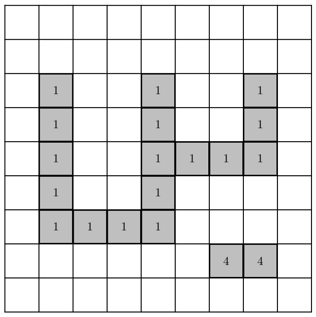


Figure 8. upper-left: binary image, upper-right: result after first pass, where the two conflict cases M1 und M2 are marked, bottom-left: merge table after the first run and look-up table after merge table resolving, bottom right: label mask after second pass.

of conflicts, the classical methods are not able to label an image in a reasonable time. Modern two pass labeling methods, as described in an article from Lacassagne and Zavidovique,[1] utilize union-find data structures, which allow even the resolution of very big equivalence tables in fractions of a second. One implementation of a two pass CCL-algorithm for video processing on an FPGA was presented in an article from Jablonski and Gorgon.[6] To reach the required performance and due to the restricted memory bandwidth, the equivalence table has to be stored in the block ram of the FPGA. Due to the restricted size of the block ram, the maximum number of conflicts and the labels have to be limited. A first analysis with a prototype model of a two pass CCL-algorithm based on union find data structures was carried out. Assuming that one memory access per clock cycle is possible, a $8192 * 8192$ pixel sized image is labeled in under 5 seconds, with an implementation running at 100 MHz. The run-time is dependent on the image content, but far more stable compared to multipass labeling.

3.3 One pass methods

One pass labeling methods as implemented by Bailey et al[7] 2008, or subsequent implementations as in Klaiber,[8] are specially designed for hardware solutions. One pass methods directly extract properties of connected components as frame box coordinates or center of gravity, without generating a label mask. One pass methods use the fact that, when only a few objects have to be labeled, the object information can be buffered in the block ram of the FPGA. The greatest advantage is that pixel stream processing is possible because, in-time conflict solution is feasible and no label mask has to be stored. In particular very high performance can be reached, since the processing of several pixels at once is possible. State of the art implementations allow parallel processing up to 32 pixels per clock cycle with a clock cycle rate over 100 MHz on a Virtex 6 FPGA.[8]

3.4 Other methods

Other CCL methods are based on region-growing or contour-tracing techniques. These kinds of methods label complete objects one after another and hence, have a very dynamic program flow. Therefore, they are not well suited for an FPGA implementation. Very interesting are high parallel methods based on mesh architecture using hundreds of processing cores, as presented in Manohar,[9] Bhattacharya[10] or Rasquinha and Ranganathan.[11] These highly parallelized methods have a revival in modern implementations on graphic processing units (GPUs). Examples can be found in Soman et al.[12] or Kalentev et al.[13]

In conclusion, two pass labeling methods based on union find data structures, appear to be the best approach for connected component labeling of large highly complex images on an FPGA. Sufficient processing performance can be reached, with the two pass method based on union find data structures. Furthermore, the method appeared to be practicaly implementable on an FPGA. The only serious challenge seemed to be the equivalence resolution. In the next section this issue is explained in detail.

4. TPSS-CCL-ALGORITHM

4.1 Constraints of the classical approach

In this section, the implementation of a two pass CCL-method is presented. As mentioned in the last section, the difficulty is the resolution of the equivalences. The complicating point is that the CCL-method should be able to label large images, without restriction in image content. Therefore on one hand, the number of possible occurring connected components can be very high. In the worst case, any pixel could be a connected component. On the other hand a tremendous amount of conflicts can occur. In the worst case scenario, a conflict occurs every second pixel[†]. Therefore, the equivalence table and the look-up table, can potentially become very large as shown in figure 9. For the classic CCL-labeling, an equivalence table, a LUT and a label mask are required. A

Image width/height in pixel	50	512	1024	2048	4096	8192	16384
Size in kBit	30	4719	20972	92275	402654	1744831	7516193
Size in MB	0.004	0.6	2.7	12	51	219	940
Bit depth of an label value	12	18	20	22	24	26	28

Figure 9. Size of the equivalence table, the look-up table and the label mask in dependence of image size

closer inspection surprisingly revealed that in the general case, the maximum size of the equivalence table equals the maximal size of the LUT as well as the size of the label mask. Even for small images, it seemed not viable to integrate the equivalence table or the look-up table in the block ram of the FPGA, even though it would have been advantageous for performance as well as for lowering system complexity. For very large images, the size of the equivalence table and the look-up table could even be a critical issue if these tables are placed in the ram. E.g. for images of the size $4096 * 4096$ pixels, 51 MB are needed for the equivalence table. Even the highest integrated FPGAs available today, have barely the resources to store such a look-up-table in the block ram.

Due to these facts, a special version of two-pass connected component labeling was developed, which needs either an equivalence table or a look-up table. It is called Two-Pass Single Storage connected component labeling (TPSS-CCL). The TPSS-CCL requires memory for the label mask, only. This is done by performing conflict resolution just in time and integrating the look-up table in the label mask.

4.2 TPSS-CCL Algorithm

4.2.1 Definitons

Let Ω be the spatial space of the image. The connectivity of two points $P, Q \in \Omega$ is represented in the following terms:

$$con(P,Q) := \begin{cases} 1 & \text{if P, Q connected} \\ 0 & \text{else} \end{cases}$$

The scanning-masks for 4-/8-connection labeling, which are shown in figure 10 are formulated in the following terms

$$M_4(P) = \{P, P_{upper}, P_{left}\}, \quad M_8(P) = \{P, P_{upper}, P_{left}, P_{upper-left}, P_{upper-right}\}$$

[†]see Appendix figure 21, worst case 4-connectivity: Diagonal-1 image, worst case 8-connectivity Diagonal-2 image

Note that if P is at the edge of an image, some points $P_{\Box}$ in the upper definition do not exist. In this case $con(P, P_{\Box})$ is set to 0 by definition. Let M be M_4 or M_8. Then

$$con_M(P) := \{Q \in M(P) \mid con(P, Q) = 1\} \subset \Omega$$

are the pixels of the label mask with central point P connected with P. In the following, for 4-connectivity M is set to M_4 and for 8-connectivity M is set to M_8.

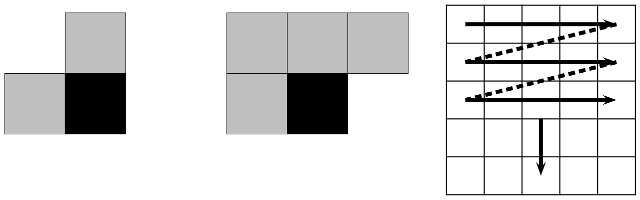


Figure 10. from left to right: TPSS-CCL mask for 4-connectivity, TPSS-CCL mask for 8-connectivity, TPSS-CCL scanning direction on image

4.2.2 First Pass

Now the first pass of the TPSS-method is presented. The temporal label mask $\mu_{label} : \Omega \to \mathbb{N}$ is created by applying the following rule to every pixel in scanning direction as shown in figure 10. Note that in this description, Ω is interpreted as a subset of $\mathbb{N}$. For example the points $P \in \Omega$ of an image could be numbered in ascending order. In the implementation, the image points are identified with their memory addresses.

Case 1 - If $con_M(P) = \emptyset$:
    $\mu_{label}(P) := P$

Case 2 - If $con_M(P) \neq \emptyset$:
    $\mu_{label}(P) := min\{head(Q) \mid Q \in con_M(P)\}$
    $\forall Q \in con_M(P) \quad \mu_{label}(Q) := min\{head(Q) \mid Q \in con_M(P)\}$

Figure 11. First pass of the TPSS-CCL method

where head is function head is a recursive function defined as:

$$head(P) := (\mu_{label}(P) == P) \ \ ? \ \ P : \ head(\mu_{label}(P))^{\ddagger}$$

4.2.3 Second Pass

The second pass is done with the rule presented in figure 12, applied on every pixel on the label mask μ_{label} obtained by the first pass. The same scanning direction as in the first pass has to be used. Furthermore, an index variable is required, which has to be set to one before starting with the second pass.

4.3 Properties

It can be proven that after the second pass, the labeling is complete and correct. Furthermore, the method is independent of the initial assignment of μ_{label} and there is no gap in the labeling indexes. This implies that if the index variable is set to one in the beginning of the second pass, the highest label equals the number of connected components of the image.

‡the expression x := a ? b : c is called ternary conditional (C-Syntax). It means x is set to a if condition b is true, else to c

```
**Initialization:** $index := 1$

**Case 1** - If $\mu_{label}(P) = P$:
    $\mu_{label}(P) := index$
    $index := index + 1$

**Case 2** - If $\mu_{label}(P) \neq P$:
    $\mu_{label}(P) := \mu_{label}(\mu_{label}(P))$
```

Figure 12. Second pass of the TPSS-CCL method

5. IMPLEMENTATION

5.1 General notes

To reach high performance, it would be advantageous to realize pixel stream processing, which has two key benefits. First, the creation of a pixel stream based processing workflow by combining various pixel stream processing filters in a row, can be done without much effort. With a pixel stream processing workflow, very high performance can be archived, because all pixel stream processing filters are executed simultaneously. The second key benefit is that image data from the ram could be fetched line by line. This is ideal for modern DDR-memory resulting in very high memory throughput. The limitation of pixel stream processing is that only the stream data is accessible without buffering. Random memory access is not possible. Pixel stream processing is therefore ideal for pixel local operations, for example a linear scaling operation. Pixel stream processing of window based filters like the Gaussian smoothing filter or the Sobel edge detection filter, can be implemented using image line buffers. Many modern FPGAs have enough block ram to buffer several image lines even if the image is large. The issue of CCL is that it involves global operation. For the determination whether two image points are connected in general, every pixel of the image has to be inspected. One example is a the shape of a big U ranging over the whole image. To distinguish that the vertical lines of the U (| |) are connected, the information of the last image line is necessary. Hence, nearly the whole image has to be read in before the connection of the two lines of the U can be determined. Therefore, pixel stream processing is not practically viable for the whole CCL process. But in the case of the TPSS-CCL method it was possible to approximately realize pixel stream processing with throughput of around one pixel per clock cycle for each of the passes. Pure pixel stream processing is only possible if there are no conflicts. For resolving the conflicts, random read/write access to the label mask is necessary. A deeper analysis with a software model of the TPSS-CCL algorithm revealed that in the most real scenarios, only a few number of conflicts occur in relation to the total amount of pixels. Therefore, a start and stop pixel stream processing approach was used for implementing the first and second pass of the TPSS-CCL algorithm. In regular operation, pixel stream processing is performed. When a conflict occurs, stream processing is stopped and the conflict is solved before resuming pixel stream processing. For more clarity, a system overview can be found in figure 13. There it is assumed, that the input image and the label mask are stored in the ram.

5.2 TPSS-CCL Algorithm in detail

In section 4.2 an abstract description of the TPSS-CCL algorithm was presented. This algorithm has to be transformed to a design which is suitable for an FPGA implementation. In figure 14 and figure 15, an explicit version of the TPSS-CCL algorithm for 4-connectivity is shown, which was used for the prototype implementation.

In the first pass, a conflict equals Case 4.2. To resolve the conflict, the head function is necessary, which needs random memory access on the label mask. In the second pass a conflict equals Case 2.3. In this case random memory access on the label mask is necessary, also. In all other cases, pure pixel stream processing has been realized. In the first and second pass, line buffering was necessary for direct access to $con(P, P_{top})$ and $\mu_{label}(P_{top})$.

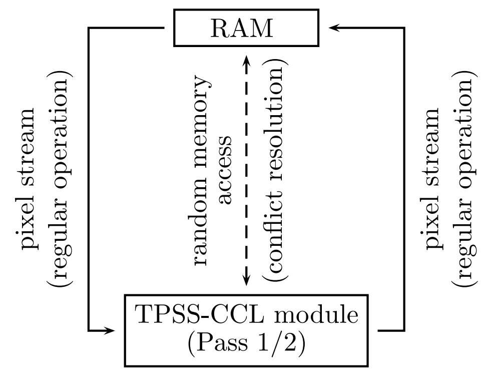

RAM
pixel stream (regular operation)
random memory access (conflict resolution)
pixel stream (regular operation)
TPSS-CCL module (Pass 1/2)

Figure 13. System overview for the first and second pass TPSS-CCL module

```
TPSS-CCL first pass (detailed, 4-connectivity)

Case 1 - If $con(P, P_{links}) == con(P, P_{oben}) == 0$
    $\mu_{label}(P) := P$

Case 2 - If $con(P, P_{left}) == 1$ und $con(P, P_{top}) == 0$
    $\mu_{label}(P) := P_{left}$

Case 3 - If $con(P, P_{left}) == 0$ und $con(P, P_{top}) == 1$
    $\mu_{label}(P) := P_{top}$

Case 4 - If $con(P, P_{left}) == con(P, P_{top}) == 1$
    Case 4.1 - If $\mu_{label}(P_{left}) == \mu_{label}(P_{top})$
        $\mu_{addr}(P) := \mu_{label}(P_{links})$
    Case 4.2 - If $\mu_{label}(P_{left}) \neq \mu_{lable}(P_{top})$
        Case 4.2.1 - If $head(P_{left}) == head(P_{top})$
            $\mu_{label}(P) := head(P_{left})$
        Case 4.2.2 - If $head(P_{left}) < head(P_{top})$
            $\mu_{label}(P) := head(P_{left})$
            $\mu_{label}(head(P_{top})) := head(P_{left})$
        Case 4.2.3 - If $head(P_{left}) > head(P_{top})$
            $\mu_{label}(P) := head(P_{top})$
            $\mu_{label}(head(P_{left})) := head(P_{top})$

with  $head(P) := (\mu_{label}(P) == P)\ ?\ P : head(\mu_{label}(P))$
```

Figure 14. Implementation of TPSS-CCL algorithm, first pass, 4-connectivity

5.3 Prototype implementation

A prototype of the TPSS-CCL method was implemented with the high-level hardware description language Handel-C,[14] currently distributed by Mentor Graphics[§]. Handel-C is a hardware description language with a C like syntax. The syntax is restricted to keywords that can be synthesized on hardware and it has no natural support for floating point operations. Hardware specific features such as parallel programming, clock management and I/O interfaces are added. Handel-C is a completely cycle based language. Assignment operations like $y = (a + b) * c$ are done in exactly one clock cycle, independent of the complexity of the operation. The complexity of the operations has indeed strong influence on the maximal achievable clock frequency. Therefore, it is important to keep complexity of the operations at a moderate level. Handel-C allows a hardware independent

[§]http://www.mentor.com (2015)

```
TPSS-CCL second pass (detailed, 4-connectivity)

Initialization: $index := 1$

Case 1 - If $\mu_{label}(P) == P$
    $\mu_{label}(P) :=$ index++
Case 2 - If $\mu_{label}(P) \neq P$
    Case 2.1 - If $\mu_{label}(P) == \mu_{label}(P_{left})$
        $\mu_{label}(P) := \mu_{label}(P_{left})$
    Case 2.2 - If $\mu_{label}(P) == \mu_{label}(P_{top})$
        $\mu_{label}(P) := \mu_{label}(P_{top})$
    Case 2.3 - Else
        $\mu_{label}(P) := \mu_{label}(\mu_{label}(P))$
```
Figure 15. Implementation of TPSS-CCL algorithm, second pass, 4-connectivity

modular development, offers debugging tools and a cycle accurate run-time analysis is possible.

For run-time analysis, four test models were created with different kinds of memory interfaces to the ram. Details can be found in figure 16. Model-1 has only one 16 Bit memory access. Therefore, saving a label takes two memory accesses for normal image sizes. Model-2 offers sufficient bandwidth for accessing one label mask entry at once. A 32 Bit access is enough for images up to 65536×65536 pixels. In most scenarios 24 Bit access is adequate. Model-3 is the model, where two independent 32 Bit ram accesses are possible. There, full stream processing is possible with reading and writing a pixel at once. Model-4 is the theoretical test model, used for the run-time analysis of a two pass algorithm. It is assumed that only memory accesses need a clock cycle, while all other operations are performed in the background. Therefore, it equals the optimal implementation if one memory access per clock cycle is possible. In figure 17, the average clocks per pixel needed for processing

Model	1	2	3	4
Random memory accesses per clock (R/W)	1 * 16 Bit	1 * 32 Bit	2 * 32 Bit	1 * 32 Bit

Figure 16. Memory interface of the TPSS-prototype models

is shown. The number of conflicts in relation to pixels of the image are shown in figure 18. The correlation between number of conflicts and run-time, is clearly obvious. Overall, the run-time is relatively stable. Even for the complex labyrinth images the run-time is raised only by around 35%. An exception is the diagonal-1 test image, which has a significantly higher run-time. The reason is that the diagonal-1 test image is the worst case scenario for 4-connectivity labeling with every second pixel a conflict. An interesting fact is that the diagonal-2 test image is the worst case of 8-connectivity labeling with also a conflict rate around 50% percent, whereas in 4-connectivity labeling no single conflict occurs.

Test image	Average clocks per pixel			
	Model 1	Model 2	Model 3	Model 4
White	7.00	4.00	2.00	4.00
Triangle	7.01	4.01	2.01	4.00
Diagonal 1	12.97	7.48	5.48	5.99
Diagonal 2	7.00	4.00	2.00	4.00
Labyrinth 1	8.19	4.65	2.65	4.35
Labyrinth 2	8.22	4.74	2.74	4.38

Figure 17. Model runtimes, synthetic test images, image size 1024*1024, 4-connectivity

Test images	White	Triangle	Diagonal 1	Diagonal 2	Labyrinth 1	Labyrinth 2
First run						
Conflicts (in %)	0.00	0.10	49.71	0.00	8.30	10.17
Second run						
Conflicts (in %)	0.00	0.10	49.71	0.00	11.54	11.54

Figure 18. Number of conflicts of the synthetic test images, image size 1024*1024 Pixel, 4-connectivity

5.4 Synthesis results

Resource utilization and maximal clock cycle frequency of the TPSS-CCL prototype implementation on three FPGA platforms are presented in figure 19. For demonstration the TPSS-CCl method was finally implemented on a Nexys 2 development board with a Spartan 3E-1200 FPGA[15] and 16 MB virtual SRAM running at 12.5 MHz.

Architecture Model	Xilinx Spartan 3E xc3s1200e-5fg320	Xilinx Spartan 6 SC6SLX75-FGG484	Xilinx Virtex 6 XC6VLX240T-FF1156
Slices	22% (1928/8672)	8% (998/11.672)	2% (976/37680)
DSP-Blocks	-	-	-
Block-Ram	21% (6/28 * 18 kBit)	3% (6/127 * 18 kBit)	1% (3/416 * 36 kBit)
Maximal frequency in MHz	69	102	185

Figure 19. FPGA resource utilization, TPSS-CCL 4-connectivity labeling, image size $1024 * 1024$ pixel, Xilinx ISE 14.7 and Virtex , Xilinx ISE, Design goal balanced

5.5 Performance

To estimate the performance of the FPGA implementation of the TPSS-CCL method, a comparison with different hardware and software implementations was carried out. The results are pictured in figure 20. For the comparison the model run-time of the FPGA Model1@12.5 MHz and Model3@100 MHz models are compared with an Intel Core2Duo@2.80 GHz (TDP 35 W) desktop CPU, with a modern ARMv7 Cortex$^{\text{TM}}$A9@667 MHz, which is part of the Xilinx Zynq®-7000 platform and the low performance ARMv6 ARM1176JZF-S@700 MHz CPU which is part of the RaspberryPi Plattform.[16] The Model1@12.5 Mhz FPGA model equals the implementation on the Nexys 2 development kit, whereas the Model3@100 MHz data are gathered with a run-time model. For the CPU performance analysis the TPSS-CCL algorithm as shown in figure 12 and 14 were implemented in standard C and executed on the target platform. The Model1@12.5 MHz has the least performance as expected. The reasons are the very low run-time frequency and 16 Bit memory interface, which results in a low cycle:pixel ratio. But still, labeling of big complex images is possible on such a low performance platform in an reasonable time if enough ram is available. The Model-3@100 MHz version, which allows pixel streamline processing, is far more powerful. It has about two times more computation performance as the ARMv7 cortex A9@667 MHz CPU. For the CPU benchmarks only one thread was used. The TPSS-CCL algorithm cannot be split into utilizing two or more threads in a favorable way. Labeling two images in parallel can be done with almost no performance loss on the ARMv7 Cortex as well as on the Intel Core2Duo platforms. Another interesting observation is that the image content has great influence on performance of the CPU implementations, too.

The conclusion of the performance analysis is that a Model-3@100 MHz implementation is powerful enough to label even big complex images in a few seconds. A single Model-3@100 MHz implementation has approximately twice the performance of an ARM Cortex A9 MPCore@667 MHz running one thread at full load. The CPU implementations perform very good, too, which nicely shows the performance of the Intel Core2Duo and the arm cortexA9 processor. For many application the performance of an ARM CortexA9 is fairly enough. Nevertheless, if very large images or a video stream have to be processed, an FPGA implementation can be a option. On hybrid FPGA/CPU platforms as the Xilinx Zynq system performance could considerable benefit from performing labeling on the FPGA part.

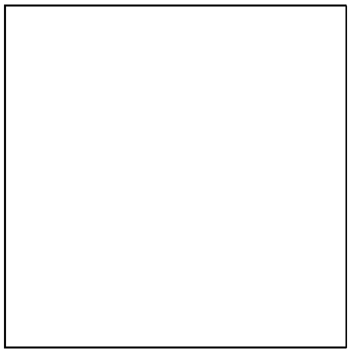
Architecture	FPGA	FPGA	ARMv6 CPU	ARMv7 CPU	x64 CPU
Model	Model-1@12.5 MHz	Model-3@100 MHz	ARM1176JZF-S@700 MHz	Cortex™ A9 MPCore™@667 MHz	Intel® ™ T9600@2.80 GHz
Test Image (1024*1024 px)	MFLOPS (FPS)				
White	1.78 (1.70)	50.02 (47.7)	6.38 (6.08)	22.65 (21.6)	170.00 (162)
Triangle	1.78 (1.70)	49.81 (47.5)	6.39 (6.09)	22.12 (21.1)	174.00 (166)
Diagonal 1	0.96 (0.92)	18.25 (17.4)	3.06 (2.92)	9.57 (9.13)	157.00 (150)
Diagonal 2	1.78 (1.70)	50.02 (47.7)	7.70 (7.35)	30.41 (29.0)	206.60 (197)
Labyrinth 1	1.55 (1.48)	37.75 (36.0)	3.79 (3.61)	14.05 (13.4)	69.73 (66.5)
Labyrinth 2	1.52 (1.45)	36.49 (34.8)	3.66 (3.49)	13.74 (13.1)	68.68 (65.5)
$\frac{\text{clocks cycle}}{\text{pixel}}$	6.70-12.96	2.00-5.48	90.83-228.62	21.93-69.67	13.55-40.77

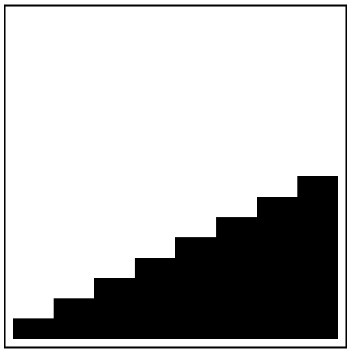
Figure 20. performance of TPSS-CCL, 4-connectivity; Software platforms: Notebook Dell Latitude E6: Intel®™ T9600@2.80 GHz, 4.0 GB DDR3, TPM 35 W, gcc 4.7.1, compiler options O3, mtune native, 64bit; RaspberryPi Model-B: ARM1176JZF-S (700 MHz), 512 MB RAM, 3.5 W, gcc, compiler options -march=armv6 -mfpu=vfp -mfloat-abi=hard; Zed Board™: Xilinx Zynq®-7000 SoC XC7Z020-1, 512 MB DDR3, arm-xilinx-linux-gnueabi-gcc (gcc version 4.8.1) -O3

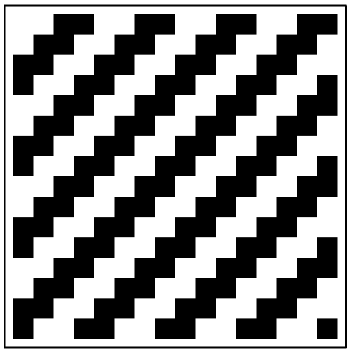
6. CONCLUSION

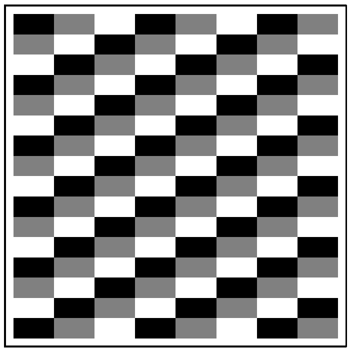
In this paper the implementation of a connected component labeling algorithm is capable of (multimodal) labeling high resolution images with no restriction to image content has been presented. With the usage of a specially developed two pass labeling algorithm labeling with a throughput of almost two clock cycles per pixel could be realized. A 100 MHz implementation is capable of labeling an 16384*16384 pixel sized image in around 6 seconds. Furthermore, a 100 MHz implementation is approximately as fast as a fully loaded ARM Cortex AP MPCore@667 MHz two core CPU, which is central part of the Xilinx Zynq platform. Implementing connected component labeling on an FPGA is therefore an option if very large complex images or a video stream has to be processed.

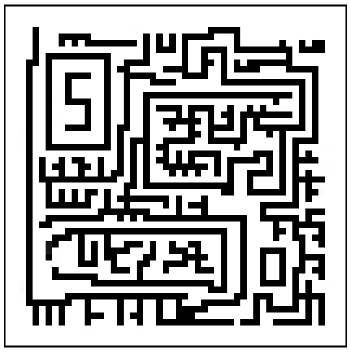
6.1 Appendix


(a) White (16*16)


(b) Triangle (16*16)


(c) Diagonal 1 (16*16)


(d) Diagonal 2 (16*16)


(e) Labyrinth 1 (50*50)

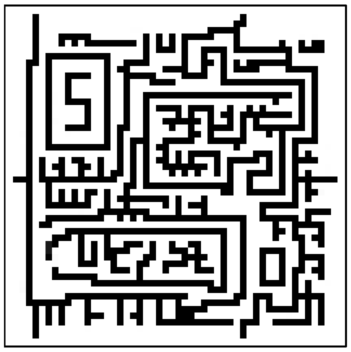
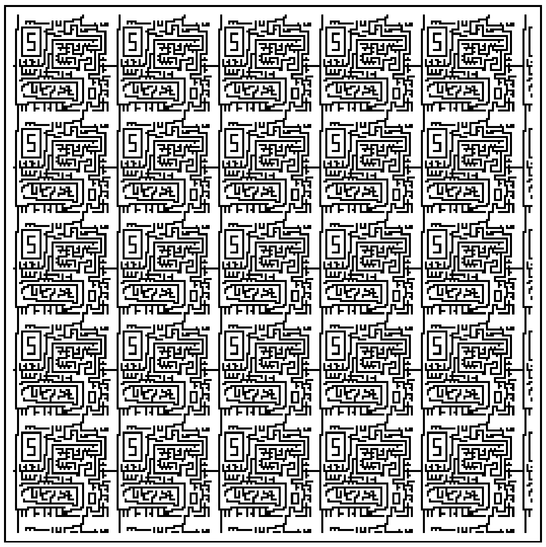

(f) Labyrinth 2 (50*50)

Figure 21. (Synthetic) test patterns, size in pixel


(a) Labyrinth 2 test image (256*256)

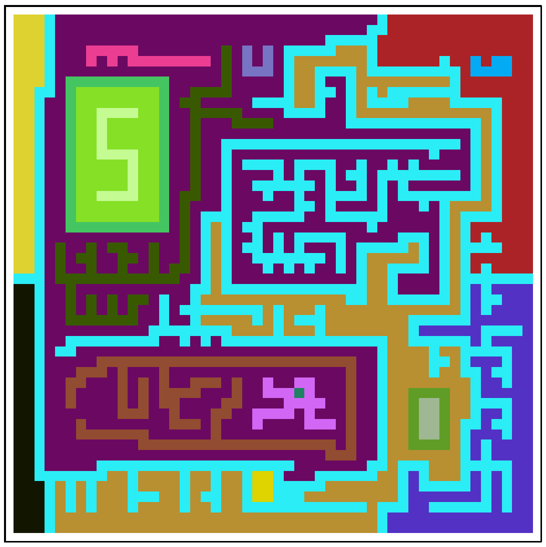

(b) Labyrinth 2 label mask of a tile (50*50)

Figure 22. left: synthetic test image, right: label mask of one tile, size in pixel

REFERENCES

[1] Lacassagne, L. and Zavidovique, B., "Light speed labeling: efficient connected component labeling on risc architectures," *Journal of Real-Time Image Processing* **6**, 117–135 (2011). 10.1007/s11554-009-0134-0.

[2] Rosenfeld, A. and Pfaltz, J., "Sequential operations in digital picture processing," *Journal of the ACM (JACM)* **13**(4), 471–494 (1966).

[3] Haralick, R., "Some neighborhood operators," in [*Real-Time Parallel Computing*], Onoe, M., Preston, Kendall, J., and Rosenfeld, A., eds., 11–35, Springer US (1981).

[4] Crookes, D. and Benkrid, K., "An fpga implementation of image component labelling," *Reconfigurable Technology: FPGAs for Computing and Applications* , 17–23 (1999).

[5] Haralick, R. and Shapiro, L., [*Computer and robot visión*], no. Bd. 1 in Computer and Robot Vision, Addison-Wesley Pub. Co. (1992).

[6] Jablonski, M. and Gorgon, M., "Handel-c implementation of classical component labelling algorithm," in [*Digital System Design, 2004. DSD 2004. Euromicro Symposium on*], 387–393, IEEE (2004).

[7] Johnston, C. and Bailey, D., "Fpga implementation of a single pass connected components algorithm," in [*4th IEEE International Symposium on Electronic Design, Test & Applications*], 228–231, IEEE (2008).

[8] Klaiber, M., Bailey, D., Ahmed, S., Baroud, Y., and Simon, S., "A high-throughput fpga architecture for parallel connected components analysis based on label reuse," in [*Field-Programmable Technology (FPT), 2013 International Conference on*], 302–305 (Dec 2013).

[9] Manohar, M. and Ramapriyan, H., "Connected component labeling of binary images on a mesh connected massively parallel processor," *Computer Vision, Graphics, and Image Processing* **45**(2), 133 – 149 (1989).

[10] Bhattacharya, P., "Connected component labeling for binary images on a reconfigurable mesh architecture," *Journal of systems architecture* **42**(4), 309–313 (1996).

[11] Rasquinha, A. and Ranganathan, N., "C3l: a chip for connected component labeling," in [*VLSI Design, 1997. Proceedings., Tenth International Conference on*], 446–450 (1997).

[12] Soman, J., Kishore, K., and Narayanan, P. J., "A fast gpu algorithm for graph connectivity," in [*Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*], 1–8 (2010).

[13] Kalentev, O., Rai, A., Kemnitz, S., and Schneider, R., "Connected component labeling on a 2d grid using {CUDA}," *Journal of Parallel and Distributed Computing* **71**(4), 615 – 620 (2011).

[14] Alston, I. and Madahar, B., "From c to netlists: hardware engineering for software engineers?," *Electronics & Communication Engineering Journal* **14**, 165–173(8) (August 2002).

[15] Corporation, X., "Spartan-3e fpga family: complete data sheet," (2013).

[16] Upton, E. and Halfacree, G., [*Raspberry Pi User Guide*], John Wiley & Sons (2013).