Optimised Single Pass Connected Components Analysis

Ni Ma
Department of ECSE
Monash University, Clayton, Australia
email: ni.ma@eng.monash.edu.au

Donald G. Bailey, Christopher T. Johnston
School of Engineering and Advanced Technology
Massey University, Palmerston North, New Zealand
D.G.Bailey@massey.ac.nz, c.t.johnston@massey.ac.nz

Abstract

Classical connected components labelling algorithms are unsuitable for real-time processing of streamed images on an FPGA because they require two passes through the image. Recently, a single-pass algorithm was proposed that avoided the need to buffer an intermediate image. In this paper, a new single pass algorithm is described that is a considerable improvement over the existing algorithms. The new algorithm reassigns and reuses labels each row to minimise the size of both the equivalence and region data tables. The optimised single-pass algorithm reduces the worst case memory requirement by over 100 times that of the original algorithm (for measuring region area), and reduces the latency to only 1 row.

1. Introduction

Connected components analysis is an important step in many image analysis and machine vision applications. There are typically four stages to such algorithms as shown in Figure 1. First the input (colour or greyscale) image is preprocessed through filtering and thresholding to segment the objects from the background. The preprocessed image is usually binary, consisting of a number of regions against a background. Next, connected components labelling is used to assign each region a unique label, enabling the individual objects to be distinguished. In the third stage, each region is processed (based on its label) to extract a set of features of the object represented by the region (for example: area, centre of gravity, bounding box, average colour or pixel value etc). In the final stage, these features are used to classify each region into one of two or more classes.

When implementing such an algorithm for real-time processing on an FPGA (Field Programmable Gate Array) the image data is streamed from the camera in a raster format. The preprocessing operations (typically filters and point operations) are ideally suited for stream-based processing without image buffering (apart from row caching for local filters), making their FPGA implementation straightforward. Unfortunately, the classical labelling algorithm [1] requires two passes through the image, requiring buffering of the intermediate image. Such buffering also introduces significant latency into the labelling algorithm.

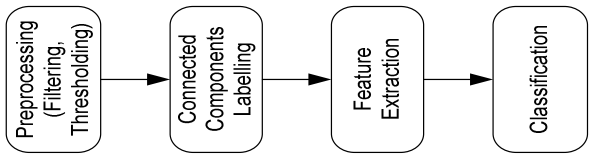

Preprocessing (Filtering, Thresholding)
Connected Components Labelling
Feature Extraction
Classification

Figure 1. Pipelined dataflow of connected components analysis

1.1. Classic connected components labelling

The classic connected components algorithm [1] requires two raster-scan passes through the image. In the first pass, a temporary label is assigned to each object pixel in the image. For each object pixel, the 4 neighbours (assuming 8-connectivity) that have already been processed are examined (see Figure 2). If none of the neighbours are labelled, the current pixel is assigned a new label. If one of the neighbours is already labelled, that label is propagated to the current pixel. Whenever two or more labels are encountered in the neighbourhood, if the labels are all the same then that label is propagated as before.

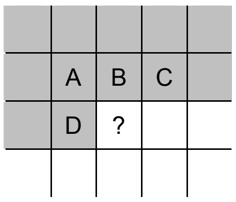


Figure 2. A label is assigned to the current pixel based on already processed neighbours

However, at the bottom of a "U" shaped object, each branch of the "U" will have a different label. Where they join at the bottom, the two branches will merge into a single object, and there will be two different labels within the neighbourhood. These two labels are now equivalent, in that they refer to the same region. In this case, one of the two labels will continue to be used, and all instances of the other label need to be replaced with the label that was retained. Since many such mergers may occur in processing an image, it is more efficient to defer the relabelling process so that all the merged labels may be changed at once. All pairs of labels corresponding to merged objects must be recorded and later resolved to determine the sets of equivalent labels corresponding to single objects.

The equivalence sets may be considered as a graph, with nodes representing the temporary labels and links representing equivalences between pairs of labels. After the first pass, all of the label equivalences are resolved. This is equivalent to finding each connected component within the equivalence graph, and assigning a new, final, label to each group of equivalent labels. In the second pass through the image the initial temporary labels are replaced by their final label.

There are many variations on how the equivalent labels are represented, and how the equivalence sets determined [2-6]. An efficient method for recording equivalences is to use a single 1D array indexed by the temporary label [5,6]. The array is initialised with the index as the content. An equivalence resulting from a merger is then represented by changing the entry for the larger label to point to the smaller.

1.2. Parallel and FPGA Implementations

While several high-speed parallel algorithms exist for connected components labelling (see for example the review in [7]), such algorithms are very resource intensive, requiring massively parallel processors. This makes them less suitable for FPGA-based implementation because the assumption is that the processors have been pre-loaded with the image data. When processing streamed images, the bandwidth bottleneck of reading in the image data destroys most of the benefits gained by massive parallelism.

Crookes, Benkrid, et al. [8,9] have implemented a resource efficient multi-pass algorithm on an FPGA. This uses very simple local processing, but requires an indeterminate number of passes to completely label the image. This makes such an algorithm unsuitable for real-time processing. The iterative nature of the algorithm also requires a buffer to hold the intermediate image between passes.

Jablonski and Gorgon [10] have implemented the classic two-pass connected component labelling on an FPGA. In doing so, they were able to take advantages of the parallelism offered by FPGA-based processing to gain considerable processing efficiencies over a standard serial algorithm. However, their two-pass algorithm still requires the image to be buffered for the second pass, and requires two clock cycles per pixel plus a small overhead for region merging.

1.3. Single Pass Algorithms

To achieve single pass operation, it is necessary to avoid the need for producing a labelled image. With "U" shaped objects, it is impossible to produce a consistently labelled image without knowing that two objects will later merge. There are two approaches to this: to look ahead to determine whether regions will merge, or to gather the region data as the image is processed during the first pass.

The look ahead approach is taken by Chang et al [11]. When an unlabelled object pixel is encountered, the object boundary is traced, assigning the label to the complete object boundary. As the raster scanning continues, the boundary label is then used to fill within the boundary with the correct, unique, label. This approach is unsuited to stream processing because the boundary tracing operation is random access, and may potentially require the entire image to be available.

The boundary coding approach can be implemented using a stream based raster scan, without necessarily looking ahead [12,13]. Such methods effectively build the boundary in sections, and combine the sections where regions merge. The boundaries can then be processed to extract the required features. While this method is amenable to processing streamed images, it is limited to obtaining shape information only, as the pixel values within the regions are no longer available. The intermediate storage for the region boundaries will generally require less memory than for the image. However, in the worst case, the storage needed for the boundary is the same as for the image.

The only other alternative is to extract the features of interest for each component or region while performing the connected components analysis [6]. This avoids the need for passing through the final labelled image to extract the region data, and therefore avoids the need for the second relabelling pass. Bailey and Johnston [14,15] have taken this approach for their FPGA implementation and have showed that the maximum processing time is 1.2 clock cycles per pixel. Their implementation effectively trades the storage needed for the frame buffer for the storage required for the data table. This may give significant resource savings for simple features (such as region area). However to handle the worst case images, the size of the table is still proportional to the image area. If it is necessary to handle the worst case, then much of the reduction in memory footprint is lost.

1.4. Paper Outline

This paper extends the approach taken by Bailey and Johnston [14,15] and optimises it to further reduce the memory requirements and latency.

The next section presents the design requirements for an efficient FPGA implementation. Section 3 identifies the weak point in the previous single-pass algorithm, and details our new approach to overcome this limitation. Section 4 demonstrates the operation of the new algorithm with a worked example. Section 5 compares the expected memory requirements of the new algorithm with both the previous algorithm and the classic two-pass algorithm. It also summarises the results of our FPGA implementation.

2. Design Requirements

For efficient implementation on an FPGA, it is desirable to minimise the resources used by an algorithm. From an image processing perspective, an important resource is the memory required to buffer any intermediate image and other data. While modern FPGAs are available with sufficient memory to hold a whole frame of image data, such devices are at the high-end of the market and are still relatively expensive. To minimise the memory requirements, it is therefore desirable to perform all of the processing on the image data as it is streamed into the FPGA.

This naturally leads to a pipelined implementation of the whole image processing algorithm, where separate processing modules are built for each operation within the chain (Figure 1). Synchronous processing of pixel-based data at the input data rate will simplify the synchronisation between the first 3 modules. Obviously the classification stage will run at a different rate because it will be operating on feature data extracted from the regions.

For real-time processing, it is also desirable to reduce the latency between data input and classification results output for two reasons. Firstly, a lower latency will tend to reduce the storage or memory requirements. Secondly, when the output is used to control an activity (for example the position of a robot or manipulator) the closed loop control is easier to design when the delays are smaller.

3. New single pass Algorithm

The biggest problem with the previous single pass algorithm [14] is that the worst case memory requirements are dependent on the area of the image. The number of entries in both the merger table and the accumulated data table both depend on the number of labels needed within the image. If the accumulated data requires many fields (for example when determining the moments of the region shapes), then the actual memory requirements may be worse than the classic two pass algorithm.

However, we can make the observation that the maximum number of regions on any one row of the image is half the width of the image. If the size of the working data structures can be reduced to this size, then considerable memory savings can be achieved [3,4]. Since the merger table and data table are indexed by region label, this requires that the labels be limited to this range as well. This can only be accomplished by recycling the labels from one row to the next. Since a labelled image is not required, the labels on each row can be allocated almost independently of the labels on previous rows as long as the labels are consistent with the connectivity as determined to that point.

Therefore, assuming that the labelling is consistent for the previous row, it is only necessary to process the objects sufficiently to obtain a consistent labelling at the end of the current row. As each pixel is only visited once, it is necessary to have data structures to record merging of regions on the current row. Since the simplest label allocation scheme is to allocate labels sequentially, starting from 1 on each new row, it will also be necessary to translate the labels assigned on the previous row to those used on the current row.

3.1. Hardware Architecture

Although the new algorithm is quite different, the hardware architecture is similar to that used by Bailey and Johnston [14,15]. It is extended by the addition of the translation table as shown in Figure 3.

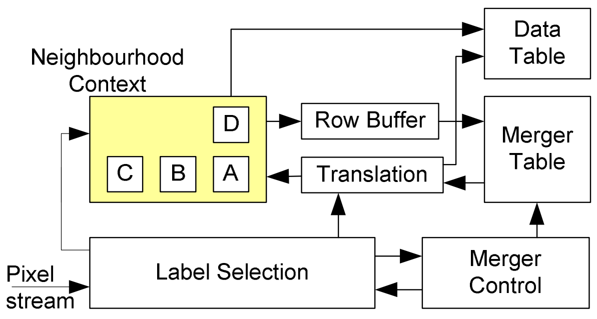


Figure 3. Connected components analysis architecture

The neighbourhood context block provides the labels of the four previously processed pixels connected to the current pixel. It is implemented in much the same manner as a window filter, with the neighbouring pixel labels stored in registers A, B, C, and D. These are shifted along each clock cycle as the window is scanned across the image. As registers A, B, and C are from the previous row, it is necessary to record both the old label (as assigned on the previous row) and equivalent label for the current row. Both labels are necessary to correctly assign a label to the current pixel.

Since the resultant labels are not saved in a temporary image, to obtain the labels from the previous row for the neighbourhood context they must be cached using a row buffer. The merger table resolves any equivalences as a result of mergers on the previous row, and the translation table translates a label allocated on the previous row to the new label assigned to that object on the current row.

3.2. Label Selection and Merger Control

The label selection block selects the label for the current pixel based on the labels of its neighbours. The label selection is based on the decision tree given in Figure 4. When a region is encountered on a row for the first time, a new label is assigned to it. All previous row labels that appear in the neighbourhood are updated with the new label. This translation is recorded in the translation table so that if that label is encountered elsewhere on the previous row, it is correctly translated to the label used on this row. If there are two different previous row labels within the neighbourhood (for example at the bottom of a "U" shaped region) then this merger is reflected in the state of the translation table. It does not need to be stored in the merger table because the data is already consistent on the current row. As one current row label represents two previous row labels after the merger, this saves the one other label which can be reused for another region later in the row.

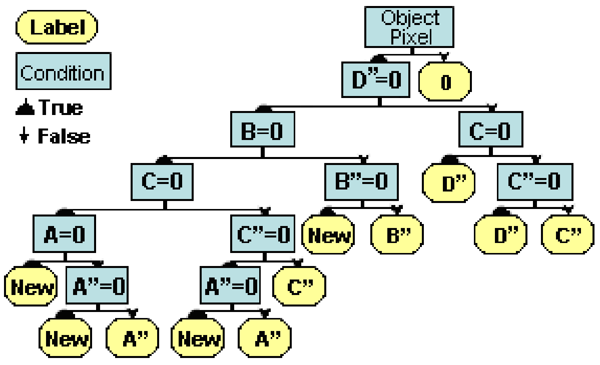


Figure 4. Label selection decision tree. A, B, and C are the labels assigned on the previous row, while A" to D" are the equivalent labels for the current row

However, if there are two different current row labels within the neighbourhood, these labels are equivalent and must be recorded in the merger table. In this case, the decision tree selects the smallest label without use of comparators. This can be proved as follows. For a merger to occur, position B must be a background pixel, and mergers only need to be considered between the neighbour pairs of D"-C" and A"-C" [2]. Consider the D"-C" pair, (points ① and ② in Figure 5). For C" to exist, the connected region must have a section which has already been processed in the current row. Therefore if D" is different, it must have been assigned after the label for C", and therefore be larger. C", as the smaller of the pair, is assigned to the current pixel. The same argument holds for the A"-C" pair (point ③ in Figure 5). This ensures the smallest possible label is used for the current label.

If all mergers are to be resolved as they are encountered, then the merger of 5 to 2 at ③ must be propagated to label 6. Furthermore, if label 2 is subsequently merged to another label further down the row then all three mergers would need to be updated. This chaining effect becomes worse as more connections are established, and must be avoided for real-time applications. Recognizing the fact that the merger table is not required until processing the next row, the label mergers are stored on a stack for processing at the end of the row, as in [14,15]. There, each pair of equivalent labels is popped off the stack in the reverse order along the row. Since the smaller label is always on the right, one pass of the stack resolves all mergers for the current row.

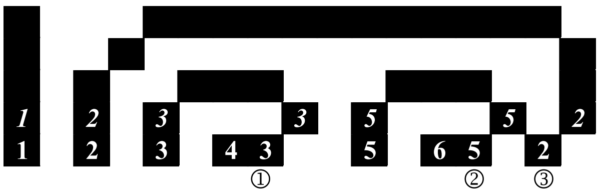


Figure 5. Example of a series of label mergers. Translated labels of the previous row pixels are represented by *italics*

Since the labels are replaced each row, two merger tables are required: one to resolve mergers on the previous row (with previous row labels), and one to record mergers for the current row. At the end of each row, the tables are swapped, with the table of new equivalences on the current row becoming the merger resolution table for the previous row, and the previous table being reset to record new mergers.

However, only a single translation table is required. It is built as the row is scanned, and at the end of the row all previous row labels will have been translated to the current row so the table may be reset. Such resetting may be performed on the fly, while building the translation table during the next row.

3.3. Data Tables

The data table accumulates the raw data from the image required for calculating the features of each connected component. Since the image data is not retained, feature data must be accumulated for each connected component as the image is scanned. Any features that may be accumulated incrementally may be measured, for example: area, moments, bounding box, average colour or pixel value. Additional logic maybe required for certain features, such as edge detection for measuring perimeter. Otherwise the only difference for different features is the width of the data table. The data table is indexed by the current pixel label, with the corresponding entry updated to reflect the inclusion of the current pixel within the region.

Whenever two regions merge, the corresponding entries in the data table are also merged [6].

As different labels are used for the previous row and current row, two data tables are required. Features are combined when regions connect, with the resulting combined features stored in the entry indexed by the current pixel label. Merged or relabelled entries are deleted from the previous row to simplify detection of completed objects; any entries not deleted will be completed regions.

This has the additional advantage that once a region is completed the features may be passed immediately to the classification processor without having to wait until the end of the image scan. This further reduces the latency with the results of each region being output at the end of the first row after the last pixel for the region is input.

4. Algorithm demonstration

The operation of the algorithm is demonstrated by processing of one row in an image, and is shown in Figure 7. For each object pixel, the steps for label selection and table manipulations are presented in a table. Changes to the tables have been highlighted.

The assigned labels for the last two rows are shown. The current row for processing is the last row of the image, and the row above is to be referred to as the previous row. Two tables are required at the start of processing the current row: the merge table for previous row labels (PM) and the data table for previous row regions (PD).

In considering the image at the end of the previous row, there are four separate regions: two single pixel regions with labels 1 and 2, one external arc with label 3, and an inner arc with labels 4 and 5. Labels belonging to a region have the same equivalent label unique to the region, which is the smallest label of that region. The equivalent label can be obtained from the merger table (PM) using the initially assigned label as the index. In this example, entries 4 and 5 have the same value 4; this indicates that pixels with initial labels 4 and 5 belong to region 4. In the data tables, existing entries are represented by "#"s. The data of label 5 has been merged with that of label 4 as part of the merger processing of the previous row. Consequently, the 5th entry in the data table for the previous row (PD) is 0.

The following tables are assumed to be initialised before the processing of a row: the data table for current row regions (CD); the translation table (T) for translating previous row labels into current row labels; the merger table for the current row labels (CM). In practise, these tables may be initialised as the current row is processed.

In the example shown in Figure 7, only the processing for the object pixels is shown. To aid understanding, the processing is described separately for each pixel. In practise, the steps take several clock cycles, and are pipelined. In the label selection section, the initial labels from the row buffer are looked up in the PM to obtain the equivalent label. These are then translated by looking up in T. The current labels within the neighbourhood context are highlighted with the heavy border. These are used to assign the initial label for the current row.

Pixel position ① demonstrates a merger of two previous row regions. A new label is given to the current pixel, the data of the regions are merged (over 3 clock cycles) and the translation table updated. Pixel position ② is a simple label translation, with the data from the previous row transferred to the new label. Pixel positions ③ and ⑤ have no neighbours so far, so new labels are given, and data recorded. Note that the cached data (DC) for ③ is not actually written to the data table because it is updated on the next clock cycle at ④. Two current row labels merge at pixel positions ④ and ⑥. The data are combined, and the label pairs are pushed on a stack for creating the merger table at the end of the row. Each time a new label is assigned, the corresponding entry in the current row merger table (CM) is initialised.

It is interesting to observe the label mergers from the "U" shapes, shown by points ① and ⑥ in Figure 7. Only labels at ⑥ need to be resolved. The "U" at ① does not have a current label for previous label 3, so the merger is handled solely by the translation table. This significantly reduces the need for label resolution over the algorithm of [14,15].

At the end of the row, the stack is processed to update CM. As each pair of labels is popped off the stack, the CM entry of the larger label is replaced by the CM entry of the smaller label. One pass of the stack is sufficient to process all label merges.

Features of complete regions are also extracted at the end of the row. The first entry in PD is still "#" at the end of processing the current row. This indicates that the region labelled 1 does not extend to the current row. The completed regions can be easily detected by reading off all non-zero entries as the PD is initialised for the next row. This differs from [14,15] which occurs at the end of frame.

5. Comparison of Methods

The most significant resource required by the algorithm are the RAMs used for the row buffer, merger tables, translation tables and data tables.

The row buffer requires a dual-port RAM, with one port used for writing the value at the input to the buffer, and the other for reading at the output. The length of the RAM is dictated by the width of the image, and the width determined by the number of labels required.

There must be one entry in each of the merger and data tables for each label used. The number of labels needed depends strongly on the size and complexity of the image being analysed, with the worst case being ¼ of the number of pixels in the image. The width of the merger table is determined by the number of labels, whereas the width of the data table depends on the complexity of the features being extracted.

The size of the stack for resolving equivalence chains depends on the complexity of the image being analysed. The worst case is reduced from half the width of the image for the previous algorithm [14] to ¼ the width of the image for the new algorithm (see Fig. 6). Each equivalence pair requires 3 clock cycles to resolve, although with a dual-port merger table this can be pipelined with a throughput of 1 equivalence pair per clock cycle [14].

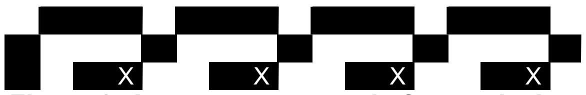


Figure 6. A worst case senario for stack size. Entries to the stack are noted by "X"

5.1. Memory Requirements

In this section the memory requirements of connected components labelling (or analysis for the single pass algorithms) are compared. It is assumed that a VGA resolution image (640x480 pixels) is being processed, obtaining the area of each region in the image. The area is one of the simplest features to extract, making the hardware easier to debug and test. Two scenarios will be compared: one will consider the worst case, and will correctly process any 640x480 image; the other will consider the requirements to process an image with 1000-1500 regions (with up to 4096 labels). The latter case will not correctly handle every possible image but will be adequate for most typical images.

In the worst case (Table 1), a 640x480 image uses 76,800 labels, requiring 17 bits for each label. For the typical case (Table 2), 4096 labels requires 12 bits for each label. The area requires up to 19 bits for each region.

For the classic connected components labelling algorithm, the whole image must be buffered between passes, and this is where most of the memory is required. A data table and chain stack are not required, but the merger table is required for storing the equivalence sets. Reducing the number of labels has a small effect on the image buffer (from the reduced bit width), but has a significant effect on size of the equivalence table.

For the original single-pass algorithm of [14,15], there are significant savings because only one row needs to be buffered rather than the whole image. The merger table is the same size as for the classic algorithm. The data table is the same length as the merger table, but is 19 bits wide. This is where most of the memory is required, and this will be made worse with more complex region data extracted. In the worst case, there is only a small improvement over the classic algorithm, but with a reduced number of labels the storage can be reduced significantly. The chain stack must be 320 deep in the worst case, but can be significantly reduced for typical images (16 allowed here). For typical images, the original single-pass algorithm requires significantly less memory than the classic algorithm.

Table 1. Worst case memory (bits)

	Classic two pass	Original single pass	Optimised single pass
Row buffer	5,222,400	10,880	5,760
Merger table	1,305,600	1,305,600	5,760
Data table		1,459,200	12,160
Chain stack		10,880	2,880
Total	6,528,000	2,786,560	26,560

Table 2. Typical memory (bits) for 4096 labels

	Classic two pass	Original single pass	Optimised single pass
Row buffer	3.686,400	7,680	4,480
Merger table	49,152	49,152	1,792
Data table		77,824	4.864
Chain stack		384	224
Total	3,735,552	135,040	11,360

The number of labels required by the optimised algorithm is only ½ the width of the image. This significantly reduces the size of both the merger and data tables. For a typical image, 128 labels have been allocated (per row), reducing the storage required for the merger and data tables further. Overall, label reuse has greatly improved the memory requirement for the single pass algorithm.

5.2. Implementation Requirements

The algorithm was developed using Handel-C with the Celoxica DK5 Design Suite. The implementation was targeted for a Celoxica RC300 evaluation board which contains a Xilinx Virtex-II XC2V6000 FPGA.

The algorithm was implemented for the worst case, processing of a 640x480 image, measuring the area of each region. No external RAM was used. See Table 3 for details. Note that this also includes a small amount of logic for generating test data and producing a VGA output for verifying correct functionality.

With the low logic cost of the design, the rest of the FPGA can be used to extract additional features, implement the object classification stage and other processes. The system operates at the at the input data rate; at this frequency, it is sufficient for processing of VGA resolution video streams at over 100 frames per second.

Table 3. Resources used

Block RAMS		4	of 144	2.8%
Slice flip flops		600	67584	0.9%
LUTs	Logic	958		
	Route through	403		
	32x1 RAM	384		
	Shift registers	12		
	Total	1757	67584	2.6%
Maximum clock frequency		40.63 MHz		

6. Conclusion

The classic two-pass connected components algorithm is not well suited for processing streamed images because it requires buffering of the intermediate image between passes. A previous single-pass algorithm removed the buffering requirement, and enables the whole algorithm to be implemented as a single streamed pipeline. That algorithm, however, still required significant storage for gathering the feature data in the worst case, although this can be reduced significantly for typical images.

A new, optimised, algorithm is proposed which significantly reduces the storage requirement by recycling labels between rows. The worst case storage requirements are no longer proportional to the area of the image, but to the image width. This enables even worst case images to be processed by a relatively modest sized FPGA. This is demonstrated by an implementation that only uses a small fraction of the resources available on an FPGA, allowing the remaining resourced to be used for other processing within the application.

A side effect of recycling the labels is that it enables completed regions to be detected at the end of the row following the region. This further improves the latency of algorithm over the original single-pass algorithm.

The approach we have taken in implementing connected components analysis is to adapt and remap an algorithm onto an FPGA rather than use the FPGA to provide acceleration of an existing algorithm. This enables the resources available on the FPGA to be used more effectively, and also allows a significantly improved implementation. In this instance, it has led to an improved algorithm that significantly reduces the memory needed, by over 100 times, and reduces the latency from the end of the frame after the object is completed to only one row after the object is complete. The optimised algorithm is suitable, not only for FPGA implementation, but for any embedded processor with limited resources.

7. Acknowledgements

The authors would like to gratefully acknowledge the Celoxica University Programme for generously providing the DK Handel-C Design Suite, and the Xilinx University Programme for providing the ISE Foundation.

8. References

[1] A. Rosenfeld and J. Pfaltz, "Sequential Operations in Digital Picture Processing", *Journal of the ACM*, **13:**(4) 471-494 (1966).

[2] K. Wu, E. Otoo, and A. Shoshani, "Optimizing connected component labelling algorithms", in *Medical Imaging 2005: Image Processing*, **SPIE 5747:** 1965-1976 (12-17 February, 2005).

[3] V. Khanna, P. Gupta, and C.J. Hwang, "Finding connected components in digital images by aggressive reuse of labels", *Image and Vision Computing*, **20:**(8) 557-568 (2002).

[4] R. Lumia, L. Shapiro, and O. Zuniga, "A new connected components algorithm for virtual memory computers", *Computer Vision, Graphics, and Image Processing*, **22:** 287-300 (1983).

[5] L. He, Y. Chao, and K. Suzuki, "A Linear-Time Two-Scan Labelling Algorithm", in *IEEE International Conference on Image Processing (ICIP 2007)*, San Antonio, Texas, **V:** 241-244 (16-19 September, 2007).

[6] D.G. Bailey, "Raster Based Region Growing", in *Proceedings of the 6th New Zealand Image Processing Workshop*, Lower Hutt, New Zealand, 21-26 (29-30 August, 1991).

[7] H.M. Alnuweiti and V.K. Prasanna, "Parallel architectures and algorithms for image component labeling", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **14:**(10) 1014-1034 (1992).

[8] K. Benkrid, D. Crookes, and A. Benkrid, "Towards a general framework for FPGA based image processing using hardware skeletons", *Parallel Computing*, **28:** 1141-1154 (2002).

[9] D. Crookes and K. Benkrid, "An FPGA Implementation of Image Component Labelling", in *Reconfigurable Technology: FPGAs for Computing and Applications*, **SPIE 3844:** 17-23 (August, 1999).

[10] M. Jablonski and M. Gorgon, "Handel-C Implementation of Classical Component Labelling Algorithm", in *2004 Euromicro Symposium on Digital System Design (DSD 2004)* Rennes, France, 387-393 (31 August - 3 September, 2004).

[11] F. Chang, C.-J. Chen, and C.-J. Lu, "A linear-time component-labeling algorithm using contour tracing technique", *Computer Vision and Image Understanding*, **93:** 206-220 (2004).

[12] E. Mandler and M.F. Oberlander, "One-pass encoding of connected components in multivalued images", in *Proceedings of the 10th International Conference on*

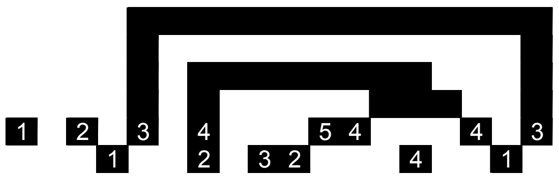
Pattern Recognition, Atlantic City, NJ, **2:** 64-69 (16-21 June, 1990).

[13] P. Zingaretti, M. Gasparroni, and L. Vecci, "Fast chain coding of region boundaries", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **20:**(4) 407-415 (1998).

[14] D.G. Bailey and C.T. Johnston, "Single Pass Connected Components Analysis", in *Image and Vision Computing New Zealand*, Hamilton, New Zealand, 282-287 (5-7 December, 2007).

[15] C.T. Johnston and D.G. Bailey, "FPGA implementation of a Single Pass Connected Components Algorithm", in *IEEE International Symposium on Electronic Design, Test and Applications (DELTA 2008)*, Hong Kong, 228-231 (23-25 January, 2008).

Binary input image; last row is current row of processing



Tables at start of row

CD	0	0	0	0	0
PD	#	#	#	#	0
T	0	0	0	0	0
CM	0	0	0	0	0
PM	1	2	3	4	4

	Label Selection			Table manipulation			
①	Row buffer After merger table After translation	2 0 3 2 0 3 0 0 0 0	New label 1	DC = CP $\oplus$ PD[2] PD[2] = 0 T[2] = 1 CM[1] = 1	DC = DC $\oplus$ PD[3] PD[3] = 0 T[3] = 1	CD[1] = DC	CD # 0 0 0 0 PD # 0 0 # 0 T 0 1 1 0 0 CM 1 0 0 0 0
②	Row buffer After merger table After translation	0 4 0 0 4 0 0 0 0 0	New label 2	DC = CP $\oplus$ PD[4] PD[4] = 0 T[4] = 2 CM[2] = 2	CD[2] = DC		CD # # 0 0 0 PD # 0 0 0 0 T 0 1 1 2 0 CM 1 2 0 0 0
③	Row buffer After merger table After translation	0 0 0 0 0 0 0 0 0 0	New label 3	DC = CP CM[3] = 3	CD[3] = DC		CD # # 0 0 0 PD # 0 0 0 0 T 0 1 1 2 0 CM 1 2 3 0 0
④	Row buffer After merger table After translation	0 0 5 0 0 4 0 0 2 3	Min label 2	DC = DC $\oplus$ CP $\oplus$ CD[2] CD[3] = 0 Stack (3,2)	CD[2] = DC		CD # # 0 0 0 PD # 0 0 0 0 T 0 1 1 2 0 CM 1 2 3 0 0
⑤	Row buffer After merger table After translation	0 0 0 0 0 0 0 0 0 0	New label 4	DC = CP CM[4] = 4	CD[4] = DC		CD # # 0 # 0 PD # 0 0 0 0 T 0 1 1 2 0 CM 1 2 3 4 0
⑥	Row buffer After merger table After translation	4 0 3 4 0 3 2 0 1 0	Min label 1	DC = CP $\oplus$ CD[2] CD[2] = 0 Stack (2,1)	DC = DC $\oplus$ CD[1]	CD[1] = DC	CD # 0 0 # 0 PD # 0 0 0 0 T 0 1 1 2 0 CM 1 2 3 4 0

At end of the row, the stack is processed update current row merger table:

Stack		CM					
			1	2	3	4	0
(2,1)	CM[2] = CM[1]		1	1	3	4	0
(3,2)	CM[3] = CM[2]		1	1	1	4	0

Figure 7. Processing of a single row: In the label selection, the current neighbourhood context is shown with heavy border. Abbreviations: CD – current row data table; DC – data cache; PD – previous row data table; T – translation table; CM – current row merger table; PM – previous row merger table; $\oplus$ – data combination operation; # – valid region data. Note that the table manipulations are pipelined over up to 3 clock cycles