CONNECTED COMPONENTS ANALYSIS OF STREAMED IMAGES

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

Ni Ma

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

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. The basic requirements of connected components analysis are investigated, and this led to a novel single-pass approach that avoids the need to buffer an intermediate image. This is further analysed to give an algorithm that is both memory efficient and has the minimum latency. The final algorithm reduces the memory by over 300 times, and reduces the latency by more than a factor of 2.

1. INTRODUCTION

Connected components labelling is an important step in many image analysis applications. There are four stages to such algorithms as shown in Fig. 1. First the input (colour or greyscale) image is filtered and thresholded to segment the objects from the background. Next, connected components labelling is used to assign each region a unique label, enabling the distinct objects to be distinguished. In the third stage, each region is processed (based on its label) to extract a set of object features (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.

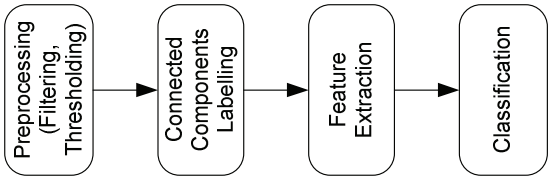

Fig. 1. Dataflow of connected components analysis

When implementing such an algorithm for real-time processing on an FPGA the image data is streamed from the camera in a raster format. The preprocessing operations (filters and point operations) are ideally suited to stream-based processing without image buffering (apart from row caching for local filters). Unfortunately, the classical connected components labelling algorithm [1] requires two passes through the image, requiring buffering of the intermediate image.

2. REQUIREMENTS

For efficient implementation, it is desirable to minimise the resources used by an algorithm. An important resource is the memory required to buffer any intermediate image and data. While high end FPGAs have sufficient memory on chip to hold a complete image, such devices are still relatively expensive. To minimise the memory required, it is therefore desirable to perform all of the processing on the image data as it is streamed into the FPGA. This leads naturally to a pipelined implementation, with separate modules for each operation within the chain (Fig. 1) and processing pixel-based data at the input data rate.

For real-time processing, it is also desirable to reduce the latency between data input and classification results output. This is particularly important when the output is used to control an activity (for example the position of a robot or manipulator), because closed loop control is easier to design when the delays are smaller.

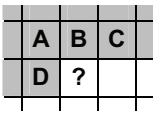

Fig. 2. A label is assigned to the current pixel based on already processed neighbours

3. CLASSIC CONNECTED COMPONENTS

The classic algorithm [1] requires two raster-scan passes through the image. In the first pass, a temporary label is assigned to each pixel. When an object pixel is encountered, the 4 neighbours that have already been processed are examined (Fig. 2). If none are labelled, a new label is assigned to the current pixel. If one neighbour is labelled, that label is propagated to the current pixel.

At the bottom of a **U** shaped object, each of the branches of the **U** will have a different label. Where they join, the two branches merge into a single object, and there will be two different labels within the neighbourhood. These two labels are 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 is retained. Since many such mergers occur in processing an image, it is more

978-1-4244-1961-6/08/$25.00 ©2008 IEEE.

efficient to defer the relabelling process so that all of the merged labels may be changed at once. All pairs of equivalent labels must be recorded for later resolution.

Equivalence sets are effectively a graph, with temporary labels (nodes) linked by equivalences. At the end of the first pass, all label equivalences are resolved by assigning a new, final, label to each set of equivalent labels. In the second pass the temporary labels are replaced by their final label. There are many variations on how the equivalent labels are stored, and how the equivalence sets are determined [2-6].

3.1. 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. When processing streamed images, the bandwidth bottleneck of reading in the image data destroys most of the benefits gained by massive parallelism.

Benkrid, et al. [8] have implemented a resource efficient multi-pass algorithm on an FPGA. This uses very simple local processing, but requires multiple passes to completely label the image. This makes such an algorithm unsuitable for real-time processing and requires buffering an intermediate image between passes.

Jablonski and Gorgon [9] implemented the classic two-pass algorithm on an FPGA. They were able to take advantage of parallelism to gain processing efficiencies over a serial algorithm. The two-pass algorithm requires buffering the image between passes, and requires two clock cycles per pixel plus a small overhead for region merging.

3.2. Single Pass Algorithms

To achieve single pass operation, it is necessary to avoid the need for producing a labelled image. With **U** shaped objects, this requires knowing that two objects will later merge. Chang et al [10] solve this by looking ahead. When an unlabelled object pixel is encountered, the object boundary is traced, assigning the label to the complete object boundary. As the raster scan continues, the boundary label is used to fill the centre of the object. This approach is unsuited to stream processing because potentially the entire image must be available for the boundary tracing operation.

Boundary coding can be implemented using a stream based raster scan, without looking ahead [11,12]. Such methods build the boundary in sections, and combine the sections where regions merge. The boundaries then require subsequent processing to extract the required features. Such methods are limited to obtaining shape information only, as the pixel values within the regions are no longer available. The intermediate storage for the boundaries will generally be less than for the image. However, in the worst case, the boundary storage is the same as for the image.

The alternative is to extract the features for each region while performing the connected components analysis [6]. This avoids the need to extract the region data from the final labelled image, and therefore avoids the second, relabelling pass. Consequently, the only intermediate storage required is a buffer of the previous row's labels.

4. OUR APPROACH

We have followed this latter approach in streaming connected components analysis [13,14]. Accumulating data during the labelling process enables all of the analysis steps to be pipelined and performed in a single pass. It is also necessary to perform the merging and equivalence resolution as the data is processed.

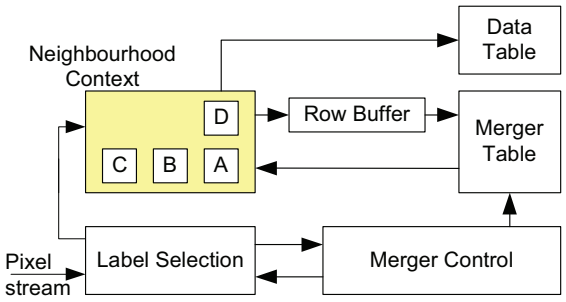

Fig. 3. Connected components analysis architecture.

4.1. Basic Architecture

The architecture of the single-pass connected components algorithm is shown in Fig. 3. The neighbourhood context block provides the labels of the previously processed pixels adjacent to the current pixel. It is implemented like a window filter, with the pixel labels shifted along each clock cycle as the window is scanned across the image. Labels from the previous row must be cached using a row buffer.

The label selection block selects the label for the current pixel based on the labels of its neighbours. This follows the same approach as the classic two pass algorithm for propagating the label. When two regions merge, the smaller of the labels is propagated. Old labels cached in the row buffer are translated to their equivalent label by the merger table before reloading into the neighbourhood context.

As the image is scanned, the data table accumulates the raw data for calculating the features of each connected component. The data table is indexed by the label of the current pixel, with the entry updated to include the current pixel within the region. Whenever two regions merge, the corresponding entries in the data table are also merged [6].

The merger control block records the equivalence by updating the merger table. The smaller label is written to the address of the larger, redundant label, so that instances of the old label from the row buffer will be replaced on the next row. Whenever a new label is created, the merger table is initialised by adding a new entry that points to itself.

When there is a series of mergers with the smaller label on the right, as illustrated in Fig. 4, each merger requires updating several equivalent labels. This cannot be completed in the single clock cycle required. These chains may be resolved with a reverse scan of the row. Pairs of equivalent labels are pushed onto a stack and at the end of the row they are popped off in reverse order for processing.

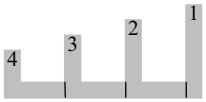

Fig. 4. Merger chain: 4=>3; 3=>2; 2=>1.

4.2. Resource and Timing Analysis

The most significant resource is the RAMs used for the row buffer, merger and data tables. The row buffer requires a dual-port RAM of length equal to the width of the image, and 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, 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 merger chain stack also depends on the complexity of the image. In the worst case 20% of the pixels result in mergers requiring stacking for equivalence resolution [13]. With a dual-port merger table, unchaining can be pipelined with a throughput of 1 equivalence pair per clock cycle [13]. The number of clock cycles required to process the image is therefore between 1.0 and 1.2 times the number of pixels in the image depending on image complexity. This is about half of the two pass approach.

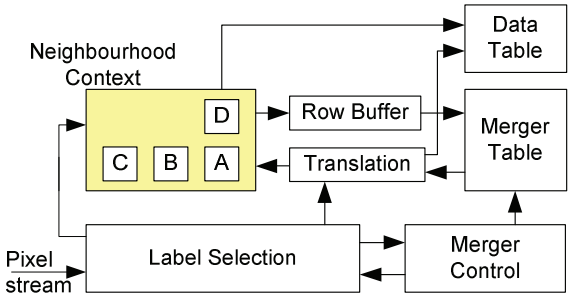

Fig. 5. Architecture of final algorithm.

4.3. Memory and Latency Optimisation

The implementation described above has traded frame buffer storage for data table storage. However to handle the worst case images, the size of the table is still proportional to the image area.

The maximum number of regions on any row of the image is ½ the width of the image. Therefore the storage requirements of both the merger and data tables may be significantly reduced by recycling the labels that are no longer required [3,4]. This also allows features of completed regions to be passed immediately to the classification processor, further reducing the latency.

Label reuse requires the regions to be renumbered each row. The equivalence table is now split into two tables (Fig. 5), with the merger table reflecting equivalences between labels on the previous row, and a translation table to update labels from the previous row to the current row. The merger and translation tables are reset and rebuilt every row. This requires two merger tables: one to handle mergers that have taken place in the previous row, and one to record new equivalences in the current row. At the end of each row, the tables are swapped. The translation table is built as the row is scanned, and may be discarded at the end of the row.

Label selection logic now also needs to assign a new label each time a region is first encountered on a row. All previous row labels in the neighbourhood are updated with the new label, and the translation recorded in the translation table. If there are two different previous row labels, then this merger is reflected in the state of the translation table, and does not need to be stored in the merger table.

A merger requires two current row labels in the neighbourhood. It can be shown that the label on the right will always be smaller. Each pair of equivalent labels is pushed on the stack for unchaining at the end of the row. The worst case stack size is now ¼ the width of the image.

The data table must be replaced each row as the labels are changed. Features of the current row are stored in a table indexed by the current row labels, with a second table for the previous row features. Merged or relabelled entries are transferred to the current table and deleted from the previous row. This simplifies detection of completed objects; any entries not deleted will be completed regions. Although two tables are required, the length of each table is only half the width of the image.

5. COMPARISON OF METHODS

It is assumed that a 1024 x 1024 pixel image is being processed, obtaining bounding box of each region in the image. Two scenarios will be compared: the worst case will correctly process any 1024 x 1024 image; the second is sufficient to process a typical image with up to 1000-2000 regions (up to 8192 labels).

In the worst case (Table 1), a 1024x1024 image requires 262,144 labels, requiring 18 bits for each label. For the typical case (Table 2), 8192 labels require 13 bits for each label. The bounding box requires 40 bits for each region.

For the classic two pass algorithm, the whole image must be buffered between passes. 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, but has a significant effect on size of the equivalence table.

The basic single-pass algorithm only buffers one row 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 40 bits wide. In the worst case, there is not much improvement over the classic algorithm, but with a reduced number of labels the storage can be reduced significantly. The chain stack must be 512 deep in the worst case, but can be significantly reduced for typical images (32 allowed here).

The number of labels required by the optimised algorithm is only ½ the width of the image, reducing the size of both the merger and data tables. For a typical image, 256 labels have been allowed (per row) reducing the storage required for the merger and data tables further. With the optimised algorithm, there are fewer mergers requiring resolution, so the stack has been reduced to 16 elements.

Table 1. Worst case memory (bits) for a 1024x1024 image.

	Classic	Single pass	Optimised
Image (row) buffer	18,874,368	18,432	9,216
Merger table	4,718,592	4,718,592	9,216
Data table		10,485,760	40,960
Chain stack		18,432	4,608
Total	23,592,960	15,241,216	63,000

Table 2. Typical memory (bits) for 8192 labels.

	Classic	Single pass	Optimised
Image (row) buffer	13,631,488	13,312	8,192
Merger table	106,496	106,496	4,096
Data table		327,680	20,480
Chain stack		832	416
Total	13,737,984	448,320	33,184

6. CONCLUSION

The classic two-pass connected components algorithm is not well suited for processing streamed images because it requires buffering the intermediate image between passes. Adapting the algorithm to a single-pass requires gathering feature data for each region in parallel with labelling, enabling a single streamed pipeline implementation.

The basic single-pass algorithm still requires significant storage in the worst case, although this can be reduced significantly for typical images. By recycling labels between rows, the worst case storage requirements are proportional to the image width rather than area. This new algorithm enables even worst case images to be processed using a modest sized FPGA. Completed regions are detected at the end of the row following the region, improving the latency over the basic single-pass algorithm.

The approach we have taken is to adapt and remap an algorithm onto an FPGA rather than use the FPGA to provide acceleration of an existing algorithm. It has reduced the memory needed by over 350 times, and improved the latency by more than a factor of 2. The improved algorithm is suitable, not only for FPGA implementation, but on any embedded processor with limited resources.

7. ACKNOWLEDGEMENTS

The authors gratefully acknowledge Celoxica University Programme for providing the Handel-C Design Suite, and Xilinx 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", *Medical Imaging 2005: Image Processing*, **SPIE 5747:** 1965-1976 (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:** 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*, San Antonio, Texas, **V:** 241-244 (2007).

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

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

[8] K. Benkrid, S. Sukhsawas, D. Crookes, and A. Benkrid, "An FPGA-Based Image Connected Component Labeller", *International Conference on Field Programmable Logic and Applications*, **LNCS 2778:** 1012-1015 (2003).

[9] M. Jablonski and M. Gorgon, "Handel-C implementation of classical component labelling algorithm", *Euromicro Symposium on Digital System Design,* 387-393 (2004).

[10] 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).

[11] E. Mandler and M.F. Oberlander, "One-pass encoding of connected components in multivalued images", *Proceedings of the 10th International Conference on Pattern Recognition*, Atlantic City, NJ, **2:** 64-69 (1990).

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

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

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