This work was published in Proceedings of the International Conference on Field-Programmable Technology (FPT 2012), pages 159 - 165 , 2012.

A Memory-Efficient Parallel Single Pass Architecture for Connected Component Labeling of Streamed Images

Michael Klaiber, Lars Rockstroh, Zhe Wang, Yousef Baroud and Sven Simon

Institute for Parallel and Distributed Systems
University of Stuttgart, Stuttgart, Germany
michael.klaiber@ipvs.uni-stuttgart.de

***Abstract*—In classical connected component labeling algorithms the image has to be scanned two times. The amount of memory required for these algorithms is at least as high as for storing a full image. By using single pass connected component labeling algorithms, the memory requirement can be reduced by one order of magnitude to only a single image row. This memory reduction which avoids the requirement of high bandwidth external memory is essential to obtain a hardware efficient implementation on FPGAs. These single pass algorithms mapped one-to-one to hardware resources on FPGAs can process only one pixel per clock cycle in the best case. In order to enhance the performance a scalable parallel memory-efficient single pass algorithm for connected component labeling is proposed. The algorithm reduces the amount of memory required by the hardware architecture by a factor of 100 or more, for typical image sizes, compared to a recently proposed parallel connected component labeling algorithm. The architecture is also able to process an image stream with high throughput without the need of buffering a full image.**

I. Introduction

The task of connected component labeling (CCL) is an important processing step in many image processing applications. It performs the task of labeling all connected image pixels in a binarized image in order to identify objects or compute certain features of an object. The throughput of the CCL can strongly influence the performance of the whole image processing system as it is one of the first complex processing steps in image processing applications. For this reason, a parallel CCL algorithm with a memory-efficient architecture is proposed here suited for high performance image processing applications. It is based on a scalable single pass CCL algorithm which is memory-efficient and therefore especially suited for FPGAs. By using the proposed architecture and algorithm it is possible to achieve a high processing throughput for performing connected component labeling of streamed images without the need of buffering a full image, which requires either much FPGA-internal memory or requires an external high-bandwitdh memory.

978-1-4673-2845-6/12/$31.00 © 2012 IEEE

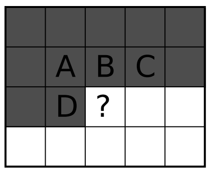

Fig. 1. Pixel neighborhood.

In the next section, the state of the art and related algorithms which cover memory-efficient and scalable CCL algorithms are presented. In Section III the proposed algorithm and the associated architecture are presented. In Section IV the proposed algorithm and architecture are discussed and experimental results are shown and compared to existing approaches in the literature.

II. Related work

The classical CCL algorithms require two scans of a binarized image and have sequential data dependencies for every pixel to be processed [1]. For storing the region labels, a memory matrix with the same dimension as the image is reserved. If a pixel belongs to the background of the image the label 0 is assigned to it. Otherwise its label is determined by the labels of the neighborhood pixels. For this decision the neighbor pixels A, B, C and D are taken into account as shown in Fig. 1.

A single pass CCL algorithm was described by Bailey et al. [2]. The proposed architecture has two drawbacks:

- Large memory requirements: The amount of memory required in a worst case scenario depends on the height and width of the image.
- Lack of parallelism: The architecture is only capable of processing one pixel per clock cycle.

In [3], [4] Ni Ma et al. improved the memory requirements of the algorithm described in [2] by reusing labels. With this improvement the amount of memory required is significantly reduced and is only dependent on the width of the image.

Kumar et al. proposed a parallel architecture for CCL [5], [6] which enhances the single pass algorithm used in [7]. In that proposal the whole image is stored in a memory in prior to processing. In order to gain a speed-up in processing, several slices of the image are processed with different CCL units independently. Therefore the CCL units fetch one line from their image slice from the memory in sequence in a round robin fashion. After processing each slice of an image, each CCL unit passes a vector describing features of regions in their image slice (not touching one of the edges) to a global FIFO memory which collects all these feature vectors. All regions touching one or both edges of an image slice are processed by a coalescing unit (CU). This unit determines if two regions of adjacent slices are connected. If they are connected the CU merges their feature vectors. For this merging process the CCL units are connected to the CU in a round-robin manner. The merging scheme relies on the fact that each edge region has a unique label in its image slice.

In [8] Lin et al. proposed a parallel architecture for CCL which relies on a global memory matrix containing labels for each image pixel. For this approach two passes for a single image are necessary which on the one hand consumes a lot of memory and therefore FPGA resources and on the other hand needs twice as much processing cycles as a single pass approach.

III. Connected Component Labeling Algorithm and Hardware Architecture

The proposed approach is an advancement of the streaming CCL algorithm described in [3]. That architecture is able to process an image in a streaming manner by processing one pixel at a time. In order to enhance the throughput, parallelism can be introduced by utilizing several parallel CCL units operating independently [5]. For this approach the image is cut into p slices, which are processed independently and the output of the parallel CCL units is merged by a suited algorithm as follows: If two regions from neighboring slices touch the border of their slices at least in one common pixel, they are recognized to be connected. This approach relies on unambiguous labeling for regions touching the slices' edges. For this reason, the approach in [5] uses the CCL algorithm described in [7], which assigns one label for each region in the image. The amount of memory required for processing an image which is of $N \times M$ pixels for this algorithm depends on the number of regions contained by the image. In the worst case an image contains up to $\frac{N}{2} \times \frac{M}{2}$ regions.

A. Scalable Architecture

In the architecture proposed in [5], the whole image is divided to p slices and saved to a memory in p correspondent continuous data blocks. The access to this memory is granted in a round-robin manner for each CCL unit. Each slice is processed by a separate CCL unit. The CCL processing units access this memory to read one line of their image slices in each round robin cycle. The results of the CCL units are merged by the coalescing unit. This coalescing unit merges regions from two adjacent slices one after another. This means if the image is cut into p slices the coalescing unit has to do in the worst case $p-1$ mergers for each processed row.

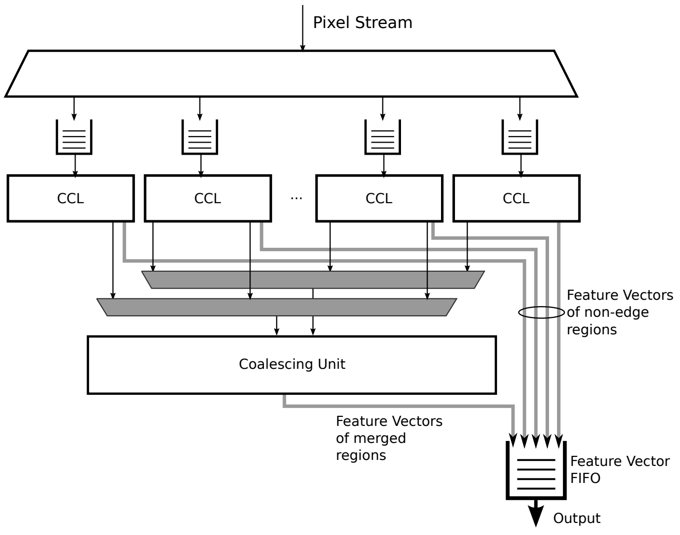
Pixel Stream
CCL
CCL
...
CCL
CCL
Feature Vectors of non-edge regions
Coalescing Unit
Feature Vectors of merged regions
Feature Vector FIFO
Output

Fig. 2. Scalable stream-based image slice processing for CCL.

As depicted in Fig. 2, the proposed architecture distributes the data pixel of one row of the image to several CCL units instead of assigning different rows to different CCL units. In this way a memory, which contains the whole image, can be avoided. This is realized by passing $n_s = \frac{N}{p}$ pixels of each row to every CCL unit. After the image has been processed by the CCL units, all feature vectors of non-edge regions are collected on the Feature Vector FIFO. The other feature vectors are passed to the coalescing unit for merging, which pushes the feature vectors also to the Feature Vector FIFO after the merging process.

The pixel stream has a higher bandwidth than the inputs of the CCL units. It is therefore necessary to buffer the pixel data before they can be passed to a CCL unit. For this purpose p buffers of the size n_s are needed. The arrangement of the buffers, the CCL units and the coalescing unit is depicted in Fig. 2.

Since the proposed stream-based slice processing architecture distributes the processing of one row to several CCL units, the coalescing unit has to merge the regions touching the slice edges of all p slices while scanning one line. To achieve this requirement a new architecture for the coalescing unit (CU) is proposed. Its main components are depicted in Fig. 3 and will be described in the following paragraphs.

The inputs of the Feature Vector Merge (FVM) block can be either connected to the CU's inputs or to the Global Data Table's (GDT) outputs. Its output is connected to the GDT. In this way both, feature vectors from the inputs and from the outputs of the GDT, can be merged.

The Global Translation Tables (GTT) have the task to address the GDT. Each of the p GTTs is responsible for the translation of the labels used in one image slice to the labels used in the GDT. They are connected to the GMT depending on which two image slices are merged. The Global Merger

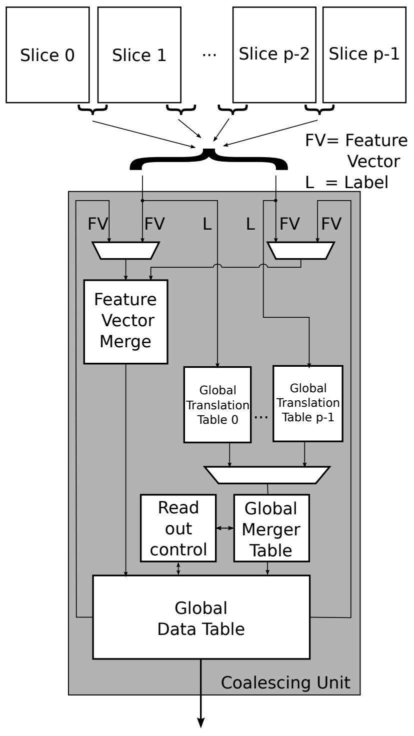

Fig. 3. Stream based coalescing unit.

Table (GMT) records mergers of two entries of the GDT and translates merged labels stored in one of the GTTs.

The merging process works as follows: All CCL units of adjacent slices are connected to the coalescing unit two at a time beginning with slice 0 and slice 1, followed by slice 1 and slice 2, until slice $p-2$ and slice $p-1$ is reached. In this way the merging of the adjacent edge pixels of the current row is performed. The CCL units connected to the CU pass the labels of the edge pixels and the associated data table entry, which contains the region's feature vector, to the CU. If two labels not being the background label 0, appear at the inputs of the CU, their feature vectors are merged and saved to the GDT of the CU. The address to which the merged feature vector is saved is determined by the GTTs. For each of the p slices of the image there is one GTT in the CU. It translates a slice label to a label that is used in the GDT. For merging processes, 3 cases can arise:

1) If two slice labels that have no entry in their associated GTT are merged, they are stored to the GDT of the CU to a new label. The GTTs of the two merged slices have to be updated to point to the new assigned label in the GDT.
2) In the case that only one slice label has an entry in its associated GTT and one does not, the feature vector of one input is merged with the GDT entry indexed by the GTT entry of the other one. After the merging process the GTTs of both labels have to point to the GDT entry of the merged feature vector.
3) If both slice labels have an entry in their GTT, two entries of the GDT have to be merged. The smaller of the two labels is used for storing the merged feature vector. This merger has to be recorded in the GMT to redirect request for an already merged feature vector of the GDT. The unused GDT entry has to be invalidated after this. In case several labels of GMT entries are building a chain, the readout control takes care of this during the readout of the GDT.

In total, p GTTs, one GDT and one GMT are needed in the coalescing unit. As each image slice can have a maximum number of M global labels for the edge regions, the GTTs should have the capacity to store M entries. To determine the size of the GDT, the maximum number of regions touching or crossing one or several slice edges have to be taken into account. This means that the GDT has to be able to store up to $2 \times (p-1) \times \frac{M}{2}$ entries in the worst case. The GMT has to have the same capacity as the GDT, since it works in the same label space as the GDT.

The pixels of the image slices are passed from the image source to the CCL units in parallel – p in each clock cycle. Therefore one row of the image consisting of N pixels is processed in $\frac{N}{p}$ clock cycles. As there are p image slices to be merged by the coalescing unit, $p-1$ mergers can occur while processing one image row. In order to be able to process these $p-1$ mergers the processing of a single merger cannot exceed more than n_{merger} cycles (as stated in inequality 1). As shown in the architecture of the coalescing unit in Fig. 3 the feature vectors at the two inputs of the coalescing unit have to pass three tables, namely GTT, GMT and the GDT. All of these tables are realized as Block-RAM and have therefore a read latency of one clock cycle. Accordingly a single merging process in this architecture takes $n_{merger} = 3$ clock cycles in order to be finished. Together with this information the maximum number of image slices one coalescing unit is able to merge can be defined by inequality 2.

$$n_{merger} \leqq \frac{N}{p \times (p-1)} \tag{1}$$

$$p \leqq \sqrt{\frac{N}{n_{merger}} + \frac{1}{4}} + \frac{1}{2} \tag{2}$$

If the desired throughput cannot be reached using p image slices according to inequality 2, the throughput can even be further increased by cascading several coalescing units in a tree using several stages. While the merging processes in the coalescing units of the first stage are taking place, the labels of the edges not being merged by a coalescing unit of the first stage have to be stored for the second stage and translated to new labels, if necessary. This can be performed until the root of the tree is reached. The coalescing unit at the root of the tree finally generates the combined feature vectors of all the image slices.

B. CCL Architecture

The new algorithm enables the parallelization mentioned in section III-A in a memory-efficient way. For applying the new algorithm, the architecture proposed in [4], [7] with

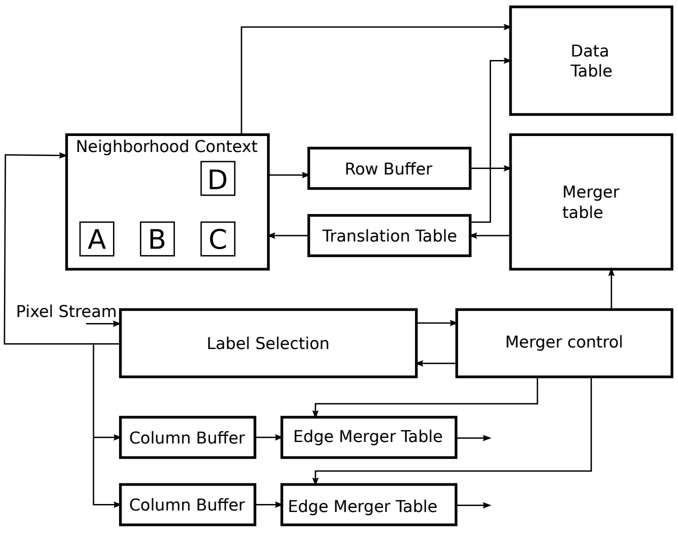

Fig. 4. Extended connected component labeling architecture.

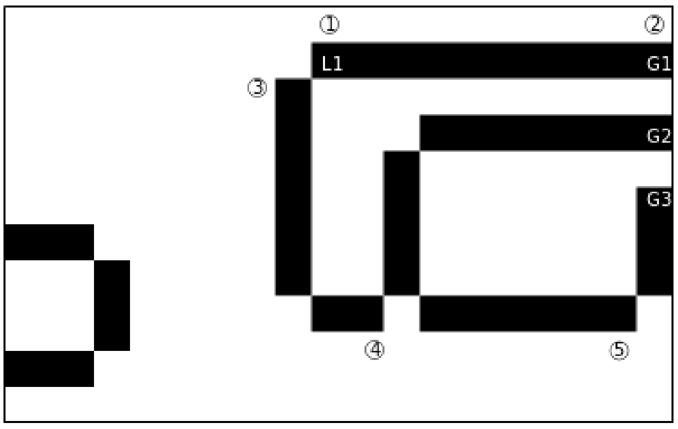

Fig. 5. Example of conversion from local slice label to global slice label and merger of global slice labels.

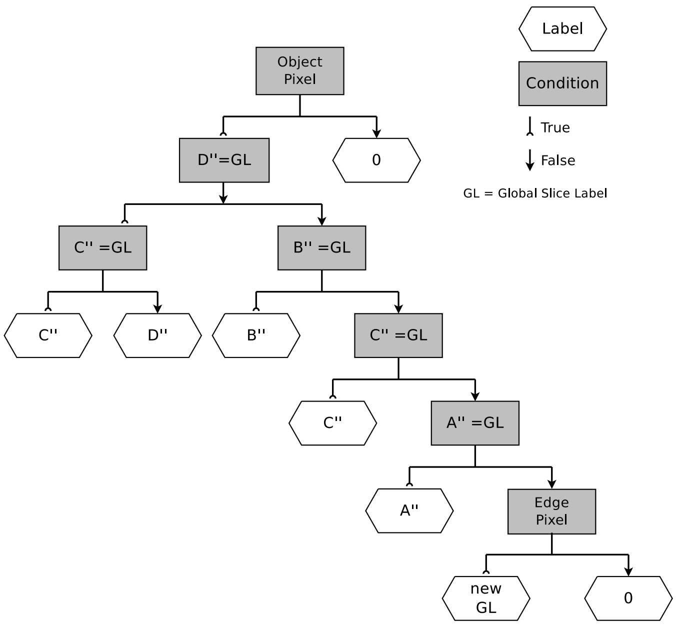

Fig. 6. Decision tree for global slice labels.

an extension proposed in [3] is used. Fig. 4 shows this architecture along with the new extensions needed for dealing with global slice labels.

The neighborhood context block consists of four registers A, B, C and D which contain the labels of the pixels located in the row above the current pixel and in the same row to the left of the current. It basically works in the same way as a window filter. Pixel data are acquired in every clock cycle from the image stream. The label of the current pixel generated by the label section block is shifted to D every clock cycle. For the generation of these labels the decision tree shown in Fig. 7 is used. This label is shifted to the row buffer. It is stored there for N cycles, where N is the number of pixels of the width of the image. It returns to the neighborhood context block after it passed the merger table and the translation table. Back in the neighborhood context block it is shifted along the registers A, B and C.

The fact that labels are reused in every row of the image leads to the case that different regions in adjacent rows of the image can have the same label. To resolve this ambiguity the translation table is introduced. It translates the labels used in the previous row to the labels used in the current row making label reuse possible.

If there are two different labels in the neighborhood context block after label translation, the regions belonging to these two labels have to be merged. This merger is recorded in the merger table. Since there can be several mergers, which have data dependencies, label pairs to be merged are pushed on a stack and evaluated at the end of the row. When the end of a row is reached, the content of the stack is read in reverse direction it was filled. In this way the merger table can be updated and all data dependencies for the mergers occurred in the current row are taken into account.

In the CCL unit the data table records the features of each region by monitoring the labels of the pixels in the neighborhood context block. Each region has one entry in the data table indexed by the region's label. Whenever a region is updated, its entry in the data table is updated as well. When two regions are merged their entries in the data tables have to be merged as well.

In order to process several parts of the image independently the image is cut into slices. Each slice is processed by one CCL unit. After all the CCL units have processed their slices, all the feature vectors of the regions are recognized and written to a memory.

Upon scanning the pixels of the image, the labels of the edge pixels are stored on column buffers that are built as cache lines of the size M. They are needed for merging the current image slice with the adjacent slices after CCL. There is one column buffer for each edge of the image slice, which is adjacent to another image slice. If the label of an edge region is written to the column buffer and this region is later merged with a

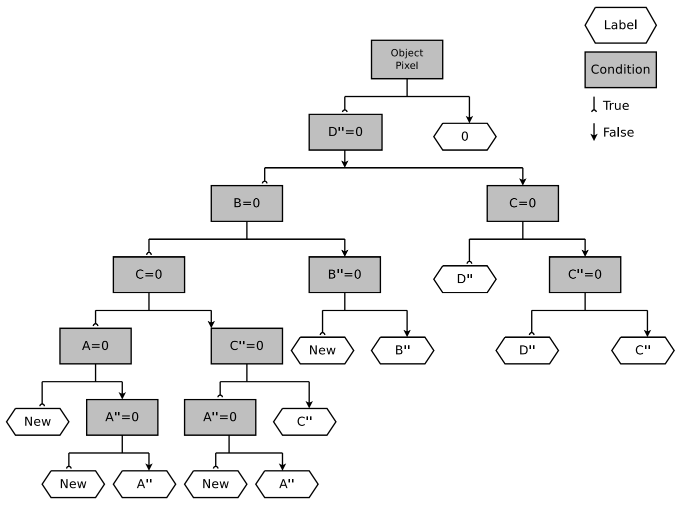

Fig. 7. Decision tree as proposed by [3]. A, B and C are the labels assigned to the regions in the previous row. A”, B”, C” and D” are the labels assigned to the regions in the current row.

different edge region, the label written to the column buffer has to be updated to the new label. Since the column buffer is built as a cache line it is not possible to access any entry but the one at the output. Therefore the edge merging table is introduced. It translates the edge labels at the output of the column buffer. At the end of each row the merger control block updates the edge merging table, with all mergers an edge label is involved in, in the current row. This is done by following the read out of the stack which is needed for the merger table. After this step it can be guaranteed that pixels of each edge region in the current image slice have the same label at the output of the edge merger table.

The reuse of labels for each row in the image, introduced in [3], has led to an enormous reduction in the memory requirement. However in section III-A it is shown that regions touching the edge of an image slice need an unambiguous label for every pixel in the region. This cannot be guaranteed by using the labeling scheme described in [3]. Therefore global slice labels are introduced. A global slice label is assigned to one region and cannot be used to mark a different region in the image slice anymore. In contrast a local slice label can be used in any row to mark a region and can be reused in every row.

The label selection block decides the current pixel’s label depending on the labels of the neighbor pixels. When a new region is detected in a row for the first time a new label is assigned. The pixel’s label can either be a local slice label or a global slice label. The local slice label which is assigned to a pixel is determined by the decision tree in Fig. 7. Simultaneously the decision whether a global slice label is assigned to the current pixel is taken by the decision tree in Fig. 6. If only one of both decision trees returns a label other than zero, the non-zero label is assigned to the current pixel. If both return a label other than zero, the global slice label is preferred and assigned to the current pixel.

Table I
REQUIRED BUFFER SIZE (BITS) FOR DIFFERENT IMAGE SIZES.

Image size	$N \times M$	1024×512	2048×1024
Kumar et al. [5]	$N \times (M + p)$	0.5×10^6 $+1024 \times p$	2.09×10^6 $+2048 \times p$
Lin et al.[8]	$log_2(\frac{N \times M}{4})$ $\times N \times M$	8.9×10^6	39.8×10^6
This work	N	1024	2048

In case a global slice label is assigned to a pixel, which has a local slice label in its neighborhood, the local slice label has to be translated to the new assigned global slice label. (point ② in Fig. 5) This translation has to be recorded in the translation table.When processing the next row the translated label appears as global slice label in the neighborhood. In case there are two global slice labels in the pixel’s neighborhood, the regions belonging to these two global slice labels have to be merged. (point ④ and ⑤ in Fig. 5) This can only be the case if there are different global slice labels in the neighborhood at position A-D or D-C [4]. When a merger among two global slice labels occurs this merger is tracked on the stack. At the end of the row the content of the merger table is updated using the entries of the stack [7]. For the update of the merger table the stack entries are read off the stack in the reverse order they were written on it. The same procedure takes place for the new introduced edge merger tables. The edge merger tables are updated simultaneously to update the merger table using the stack entries.

The distinction between local slice labels and global slice labels is realized by separating the address space of the data table, the merger table and the translation table in two parts. The upper bound of local slice labels is given by $n_{LL} = \frac{N}{2}$, as this is the maximum number of the number of regions which can appear in a row. Therefore the addresses from 1 to n_{LL} are reserved for local slice labels. A new global slice label is generated whenever an object pixel touches an edge of the image slice and does not touch another pixel with a different global slice label. For this reason the upper bound for the number of global slice labels is $\frac{M}{2}$ for each edge of the two edges of an image slice, i.e., $n_{GL} = \frac{M}{2} \times 2$. The addresses for the global slice labels range from $n_{LL} + 1$ to $n_{LL} + n_{GL} + 1$.

IV. Experimental Results and Discussion

The memory requirements for the architectures in [3], [4], [5] as well as the proposed architecture depend on the number of region labels (global slice and local slice labels) used in a single CCL unit. This is because most of the tables contained in the CCL units are indexed by the region labels.

Table II compares the number of region labels necessary in the worst case for each proposal. The architectures in [2], [5], [8] need $\frac{N \times M}{4}$ labels in the worst case. This was improved by [3] to $\frac{N}{2}$, but the image cannot be processed in several slices. The proposed architecture requires $\frac{N}{2}$ labels for local slice labels and $\frac{M}{2} \times 2 \times (p - 1)$ labels for global slice labels.

Parallel processing of several pixels of the image at the same time is only possible with the architectures in [5], [8] and the

Table II
COMPARISON OF REQUIRED NUMBER OF LABELS FOR DIFFERENT ALGORITHMS FOR A WORST CASE IMAGE.

Author	# slices	required labels for images of size			
		$N \times M$ pixels	1024×512 pixels	2048×1024 pixels	3072×2048 pixels
Bailey et al. [2]	1	$\frac{N \times M}{4}$	131072	524288	1.57×10^6
Ni Ma et al.[3]	1	$\frac{N}{2}$	512	1024	1536
Kumar et al.[5]	$2, 4, 8$	$\frac{N \times M}{4}$	131072	524288	1.57×10^6
	p	$\frac{N \times M}{4}$	131072	524288	1.57×10^6
Lin et al. [8]	p	$\frac{N \times M}{4}$	131072	524288	1.57×10^6
This work	2	$\frac{N}{2} + M$	1024	2048	3584
	4	$\frac{N}{2} + M \times 3$	2048	4096	7680
	8	$\frac{N}{2} + M \times 7$	4096	8192	15872
	p	$\frac{N}{2} + \frac{M}{2} \times 2 \times (p-1)$			

proposed architecture. To compare the memory requirements of these algorithms two criteria have to be taken into account: the amount of memory necessary for buffering image data and the amount of memory required for performing CCL.

The required memory for CCL in [5] depends on image width and height, since it is an enhancement of the algorithm in [7], the memory requirement of which depends on the image size. By introducing local and global slice labels the proposed algorithm and architecture reduced the required memory for CCL by a factor of more than 100.

Table I shows the number of required memory, which is needed for buffering in prior to processing. For the approach in [5] and [8] the whole image has to be buffered. When dealing with high resolution images this requires either a lot of FPGA resources or the usage of an external memory. In the proposed approach only one row of the image has to be stored in prior to processing, which reduces the buffer depending on the image size by a factor of more than 100.

For evaluation of the proposed architecture and algorithm the CCL unit from [3] together with the extensions proposed in section III-B and the proposed coalescing unit were described in VHDL. The synthesis results for the Xilinx Virtex 6 XC6VLX240T show that the resource utilization of the CCL units are similar to [3] except that the operating frequency here is in the range of 170 MHz depending on the image size. In [3] this architecture reached an operating frequency of 40 MHz, which is most likely from the use of an older FPGA technology. Therefore only the results of the coalescing unit are discussed. The resource utilization for the coalescing unit is shown in table III. This table gives an overview of the number of registers, look-up tables (LUTs) and block RAMs (BRAMs) required for realizing different variations of the coalescing unit merging images from 0.5 to 6 megapixels with different numbers of image slices. The table also gives the maximum operating frequency possible for the target device Xilinx Virtex 6 XC6VLX240T. By using the maximum frequency f_{max} the throughput of the coalescing unit can be calculated using equation 3. The scalability of the FPGA utilization for different number of concurrently processed image slices is shown in Fig. 8. The utilization of the coalescing unit scales approximately linear with the

Table III
RESOURCE UTILIZATION FOR COALESCING UNIT PROCESSING AN IMAGE OF 2048×1024 PIXELS
TARGET DEVICE: XILINX VIRTEX 6 XC6VLX240T SPEEDGRADE -2

$\#$ *image slices*	*Registers*	*LUTs*	*BRAMs*	f_{max} $[MHz]$
1024×512 pixels				
2	1961	2729	7	200.18
4	3128	4675	12	200.18
8	5439	8599	21	182.81
16	10060	16531	45	170.99
2048×1024 pixels				
2	3539	4908	12	199.26
4	5743	8435	22	199.26
8	10126	15538	48	172.71
16	18899	30178	95	172.20
3072×2048 pixels				
2	6141	8799	23	173.40
4	10403	15404	49	171.50
8	18910	28255	101	171.00
16	35919	54723	203	170.51
32	69923	104799	410	143.122
	of 301k	of 150k	of 416	

number of processed image slices for the used image sizes. Only the operating frequency decreases, the more the FPGA device is utilized. This also influences the throughput of the coalescing unit, which is depicted in Fig. 9. This figure shows the utilization of the target FPGA device for different image sizes and different number of processed image slices. In order to show the utilization the usage of block RAM was used since it is the most critical resource in this architecture. The utilization scales with processing throughput approximately linear in the examined area. Furthermore it is possible to merge up to 32 concurrently processed image slices in real-time and reach a processing throughput of more than 4.5 GPixels per second using the target FPGA.

$$T = f_{max} \times p \tag{3}$$

In Table IV the performance of the implementation of the proposed architecture for performing connected component labeling is compared to other hardware implementations. It is obvious that the implementation of the proposed architecture is superior in terms of processing throughput due to the high

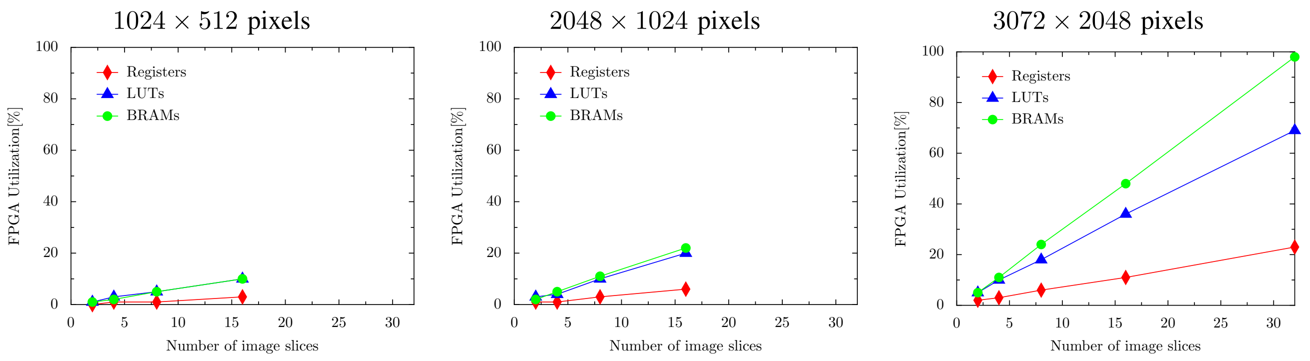

Fig. 8. FPGA resource utilization for coalescing unit for different image sizes.

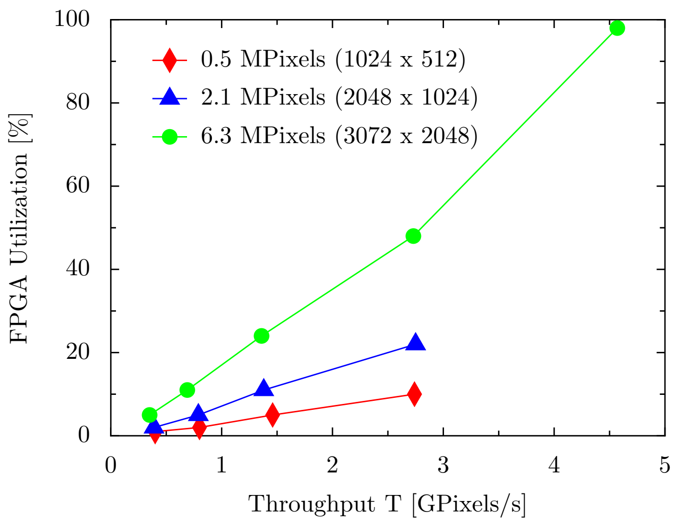

Fig. 9. Throughput of the coalescing unit for different image sizes.

Table IV
COMPARISON WITH OTHER HARDWARE IMPLEMENTATIONS.

	Parallelism $[\frac{Pixels}{cycle}]$	f_{max} $[MHz]$	Technology	Throughput $[\frac{GPixels}{s}]$
Bailey et al. [2], [7]	1	N/A	Xilinx Spartan-II	N/A
Ma et al.[3]	1	40.63	Xilinx Virtex II	0.04
Kumar et al.[5]	2	100	Xilinx Virtex 5	0.2
	4			0.4
	6			0.6
Lin et al. [8]	4	100	0.35 um	0.4
This work	4	171.5	Xilinx Virtex 6	0.7
	8	171.0		1.3
	16	170.5		2.7
	32	143.0		4.5

degree of parallelism of 32 pixels per cycle and more possible by using the proposed algorithm.

V. CONCLUSION

In classical connected component labeling (CCL) algorithms the memory requirement is at least that of a full image. The proposed parallel CCL algorithm and architecture reduce the amount of memory significantly and enable parallelism at the same time. Even when compared to sliced parallel single pass CCL algorithms and architectures, the amount of memory required is reduced by a factor of 100 or more for typical image sizes. This enables memory-efficient CCL processing of images containing the maximum number of regions in the worst case. The processing throughput of the proposed architecture can be increased compared to other architectures by the number of concurrently processed image slices and can reach on a Xilinx Virtex 6 240T up to 4.5 GPixels per second. Compared to other architectures which process sliced images the proposed architecture is suited for stream processing which is a necessity for real-time image and video processing.

ACKNOWLEDGEMENTS

The authors would like to thank the German Research Foundation (DFG) for the financial support. This work has been carried out within the research project Si 586 7/1 which belongs to the priority program DFG-SPP 1423 "Prozess-Spray".

REFERENCES

[1] A. Rosenfeld and J. L. Pfaltz, "Sequential operations in digital picture processing," *J. ACM*, vol. 13, pp. 471–494, October 1966.
[2] D. Bailey and C. Johnston, "Single pass connected components analysis," in *Proceedings of Image and Vision Computing New Zealand 2007*, december 2007, pp. 282–287.
[3] N. Ma, D. Bailey, and C. Johnston, "Optimised single pass connected components analysis," in *ICECE Technology, 2008. FPT 2008. International Conference on*, dec. 2008, pp. 185 –192.
[4] D. Bailey, C. Johnston, and N. Ma, "Connected components analysis of streamed images," in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, sept. 2008, pp. 679 –682.
[5] V. Kumar, K. Irick, A. Al Maashri, and N. Vijaykrishnan, "A scalable bandwidth aware architecture for connected component labeling," in *VLSI (ISVLSI), 2010 IEEE Computer Society Annual Symposium on*, july 2010, pp. 116 –121.
[6] V. S. Kumar, K. Irick, A. A. Maashri, and V. Narayanan, "A scalable bandwidth-aware architecture for connected component labeling," in *VLSI 2010 Annual Symposium*, ser. Lecture Notes in Electrical Engineering, N. Voros, A. Mukherjee, N. Sklavos, K. Masselos, and M. Huebner, Eds. Springer Netherlands, 2011, vol. 105, pp. 133–149.
[7] C. Johnston and D. Bailey, "Fpga implementation of a single pass connected components algorithm," in *Electronic Design, Test and Applications, 2008. DELTA 2008. 4th IEEE International Symposium on*, jan. 2008, pp. 228 –231.
[8] C.-Y. Lin, S.-Y. Li, and T.-H. Tsai, "A scalable parallel hardware architecture for connected component labeling," in *Image Processing (ICIP), 2010 17th IEEE International Conference on*, sept. 2010, pp. 3753 –3756.