Journal of Real-Time Image Processing manuscript No.
(will be inserted by the editor)

Michael J. Klaiber · Donald G. Bailey · Sven Simon

A Single Cycle Parallel Multi-Slice Connected Components Analysis Hardware Architecture

Received: 13.1.2016 / Revised: 18.4.2016 / Published: 20.6.2016

Abstract In this paper, a memory efficient architecture for single-pass connected components analysis suited for high throughput embedded image processing systems is proposed which achieves a speedup by partitioning the image into slices. Although global data dependencies of image segments spanning several image slices exist, a temporal and spatial local algorithm is proposed, together with a suited FPGA hardware architecture processing pixel data at low latency. The low latency of the proposed architecture allows reuse of labels associated with the image objects. This reduces the amount of memory by a factor of more than 5 in the considered implementations which is a significant contribution since memory is a critical resource in embedded image processing on FPGAs. Therefore, a significantly higher bandwidth of pixel data can be processed with this architecture compared to the state-of-the-art architectures using the same amount of hardware resources.

1 Introduction

Connected component analysis (CCA) is a major step in many image processing systems. It has the task of detecting pixels forming an image component and extracting its feature vectors (FV) after the image has been acquired and converted from either a colour or a greyscale representation to a binary image using a suitable segmentation method. There are several variants of CCA from multi-pass variants to two-pass variants [13] to modern single-pass algorithms. The class of single-pass CCA algorithms has the major advantage of being able to process the image without the need of storing intermediate labels in a memory matrix of the size of the image. Most modern CCA algorithms apply the union-find algorithm and use a forest data structure in order to keep a record of and merge different image segments. Union-find algorithms have been studied and evaluated extensively [15, 6, 5] and are also a foundation for the proposed algorithm and architecture.

The recent need for systems capable of processing pixel streams of high resolution images with high frame rates requires processing several pixels in parallel in order to achieve the desired throughputs. These high throughputs can be achieved in dedicated hardware architectures for CCA designed for field-programmable gate arrays (FPGAs) processing several pixels simultaneously to enable the extraction of feature vectors such as the size, area, etc. of image components without the need of storing the complete image in a memory.

There are several hardware architectures for CCA which can process up to one pixel per clock cycle [7, 4, 12, 3]. For these architectures the processing throughput is limited by the maximum clock frequency on the FPGA. Architectures capable of processing a higher throughput apply different acceleration techniques ranging from processing of a run-length encoded image stream [16, 2, 17] to parallel processing of pixels from different image rows [11] to processing images in parallel using slice processing [10, 8]. These techniques increase memory requirements and require additional processing at the end of the frame, adversely affecting latency.

The hardware architecture for parallel connected component analysis proposed in this paper is based on the architecture in [12] where one image pixel is processed per clock cycle with low memory cost and low latency, and [8] where several image pixels are processed in parallel by dividing the image into vertical slices and processing the slices in parallel. In [8] components spanning multiple slices are merged at the cost of a higher memory requirements and increased latency. The goal is to process several pixels in parallel in order to achieve a maximum throughput for processing a pixel stream. Therefore, the image is processed using an extended image processing unit similar to [4] for each image slice to detect the rela-

Michael J. Klaiber, Sven Simon
University of Stuttgart, Stuttgart, Germany
E-mail: firstname.surname@ipvs.uni-stuttgart.de

Donald G. Bailey
Massey University, Palmerston North, New Zealand
E-mail: D.G.Bailey@massey.ac.nz

tionship to neighbour slices which are saved during processing of the image and then later evaluated at the end of the image. A large memory is required to keep track of the relations between segments spanning more than one image slice. This limits a higher processing throughput even on state-of-the-art FPGAs because of a lack of memory resources. Furthermore, the processing latency for border segments is increased because they are processed at the end of the image.

In Figure 1 the proposed broad architecture for parallel CCA is depicted. The proposed FPGA architecture gains a high-throughput by partitioning the image into several image slices which are processed in parallel. The image distribution unit divides the input pixel stream to p pixel streams (one per image slice) which are provided to the slice processing units simultaneously and in raster scan order for each slice. The slice processing units (SPUs) extract feature vectors of their respective image slice and detect connected components that span multiple image slices. In addition to the extracted feature vectors and the associated global labels, the SPUs send instructions on which global labels belong to the same connected component to the coalescing unit (CU). These instructions are inter slice mergers (ISMs) and slice mergers (SMs) containing the global label information, which are introduced in Section 2.

This paper is an extended, more detailed version of the conference publication [9] providing more information on the proposed algorithm and the results. In the following section, the parallel CCA algorithm is described. The high-throughput hardware architecture is presented in Section 3. This is followed by an analysis of the algorithm and the architecture and the presentation of the experimental results in Section 4.

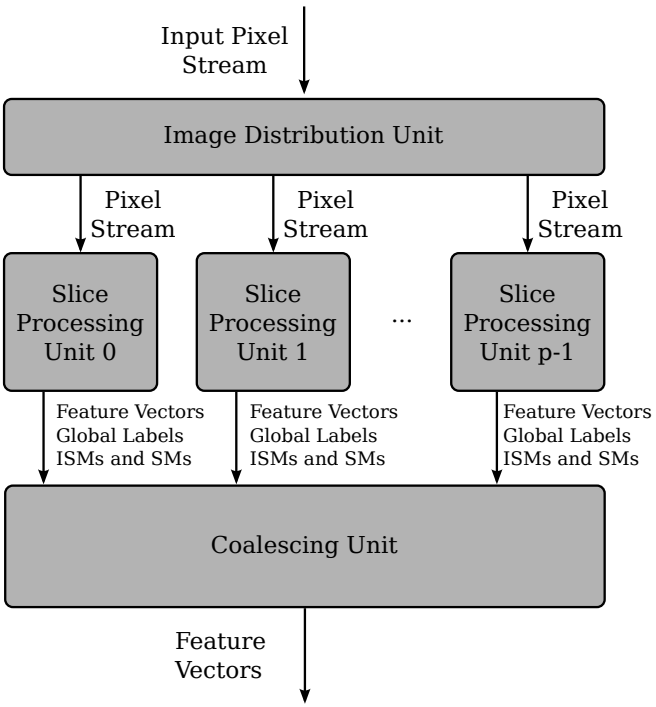

Fig. 1 Overview of a hardware architecture for parallel CCA.

2 Algorithm for parallel CCA

In this section the proposed algorithm for parallel connected components analysis is explained on an abstract level. In order to process several image pixels simultaneously in parallel, the image is divided into several vertical image slices, with each slice processed in parallel by separate pixel processing instances. Each image slice is treated as a separate image, while an additional processing instance collects information on the relationship between connected components spanning multiple image slices. If an image segment does not touch one of the slice borders, no further processing is necessary after the image segment's end is detected. This kind of segment is referred to as a local image component, because it exists only within one image slice. Whenever an image segment touches one of the slice borders, that connected component may span multiple image slices. In this case further processing is necessary to merge the feature vectors associated with those from the adjacent segments. The relationship between the local image segments and global image components is represented in a graph referred to as the global segment graph (GSG) which has to be composed during processing of the image slices. It shows the relation between global image components, global image segments and local image segments as a forest structure, in which each global image component, each global image segment and each local image segment is represented by a vertex pointing either to a parent vertex or to itself, indicating that it is the root. Since the vertical image slices are processed in raster scan order, it is possible for different global image segments to first be recognised as individual global image components, and then joined later. In this case their vertices are associated with the same connected component in I, hence their tree structures in the GSG have to be joined.

Figure 2 shows an example of an image partitioned into two image slices and its GSG. Since all of the image segments in image slice 1 and image slice 2 in Figure 2 are part of a single image component, all of the local image segments are part of a single global image component.

Due to the raster scan order, two global image segments are recognised at first, which are combined further down in the image into one global image component. The union-find algorithm is used to combine the vertices of several global image segments in a tree structure. The two basic operations for this, union and find, can be applied to the connected components problem as follows: when two global segments are merged, the find operation is used to determine the root vertices of both global image segments. After having found the roots of both tree

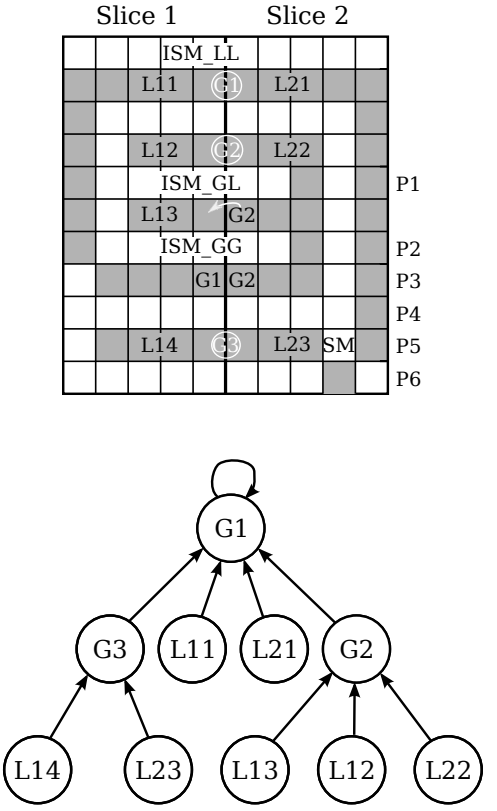

Fig. 2 Global segment graph for an example image separated to several image slices.

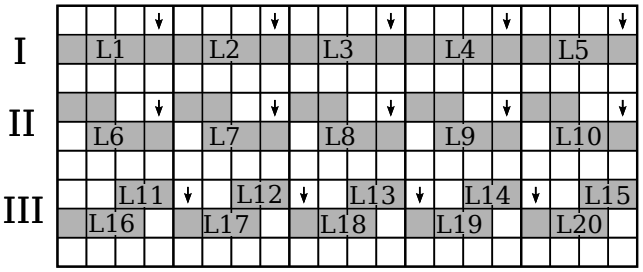

Fig. 3 Pixel contact types between neighbour slices

structures, one is merged with the other using the union operation. Additionally, path compression [14] is applied in order to minimise the cost of find operations.

Whenever an object pixel at the border of an image slice is adjacent to an object pixel (8-neighbourhood) of its neighbour slice, an inter slice merger (ISM) is detected which connects the two associated local labels. An ISM of two segments which do not yet have a global label requires the creation of a new global vertex to represent the new global image segment. Both local segments are then linked to the new global segment. If one of the local segments already has a global label, the local image segment without a global label is connected to the global vertex of the neighbouring image segment. If both local image segments already have global labels with different roots, it is necessary to merge these. These three different types of ISM operations are called ISM_LL, ISM_GL and ISM_GG. There is a fourth type of global merger, which occurs when two local segments merge within an image slice that have different global labels. These global labels belong to the same global image segment, therefore, their root vertices in the GSG have to be unified. A merger operation of this type is referred to as a slice merger (SM). The image in Figure 2 illustrates these four different merger types.

Due to 8-neighbour connectivity, ISMs can occur either horizontally or diagonally. These possible combinations are shown in Figure 3 together with the associated global segment graphs in Figure 4. The horizontal ISMs shown in Figure 3.I. are processed in raster scan order in parallel, and all the rightmost pixels of the currently processed image row at the same time. The local image segments are, therefore, part of ISMs which are detected at the same time and signalled to the neighbour slices. For this reason the vertices of the local image segments are only children of the global vertices involved in their ISMs. Global vertices of simultaneous ISMs belonging to the same global image segment have to be merged afterwards. The connections shown in Figure 3.I results in a tree structure where the local vertices are children of their associated global vertices and the global vertices are also connected to each other as shown in Figure 4.I. For diagonal connections further two cases have to be distinguished for diagonal ISMs: either the rightmost pixel of the current row has to be merged to the leftmost pixel of previous row of the right neighbour slice, or the leftmost pixel of the current row has to be merged to the rightmost pixel of the previous row of the left neighbour slice. These two possibilities are illustrated in Figure 3.II and 3.III. For a type II diagonal ISM the procedure is the same as in type I. In a type III diagonal merger, the local vertices are connected to the associated global vertices during the ISM at the start of the row. The global vertices are then later connected via an SM merger. The GSG for the example in Figure 3.III is divided into two steps: in Figure 4.III first the graph after the ISM has been carried out is shown, where the local vertices are connected to their global vertices. In the second graph the connection among the vertices after the SM mergers can be seen.

The global segment graph (GSG) keeps a record of the relation between the individual image segments in different image slices, but does not keep a record when local and global image segments are finished. A global image segment spanning several slices is finished when all of its associated local image segments are finished. In order to keep track of this, an additional counter,

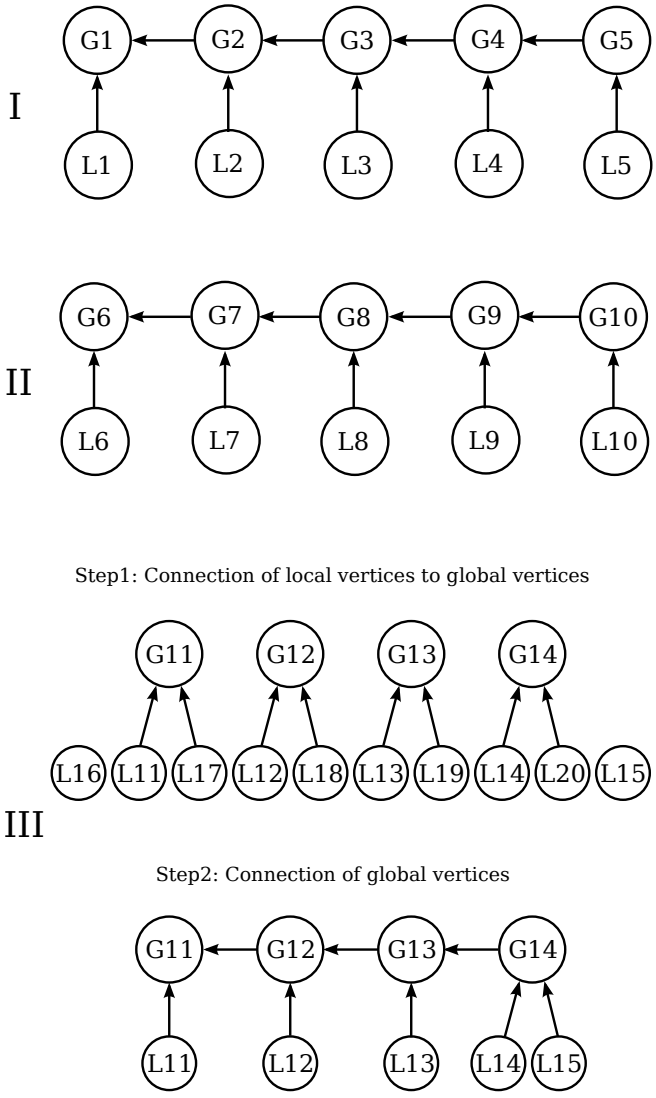

Fig. 4 Global segment graphs for the different cases in Figure 3.

referred to as child counter (CC), is assigned to each global vertex in the GSG to count the number of direct children vertices.

The CC of a global vertex is updated on every global merger, either ISM or SM. For the different merger types the CCs have to be changed as follows: for an ISM_LL merger the CC of the newly created global vertex has to be initialised with two, because two local image segments are pointing to it. It has to be incremented for an ISM_GL merger, because another global image segment is pointing to it. The two involved global vertices have to be unified at the occurrence of an ISM_GG. The root of one global vertex becomes a child of the other one. Therefore the child counter of the new root has to be incremented. For an SM merger, two local vertices are merged, which are pointing to different global vertices. In this merger, two sub-segments of one local image segment are merged. Therefore, the root vertices of the associated global vertices have to be unified and one of the global vertices loses a local vertex, i.e. its CC is decremented by one.

When a local image segment is finished, the CC of its parent's vertex is decremented. If it reaches zero, all of its constituent segments are completed and no more image segments point to that vertex. Its feature vector is, therefore, combined with the feature vector associated with its parent vertex. As a result, this global vertex and its feature vector is no longer required, and can be removed, and the CC of its parent has to be decremented. This process is repeated until the last local image segment of the global image component is finished, which results in the root vertex CC becoming zero. At this point, all local image segments of the global image segment are finished, and therefore, the global image segment is finished as well.

To outline the functionality of this algorithm, the procedure of processing two image slices and merging their local image segments on the fly is explained in the following. This is illustrated in the example image in Figure 5 which shows a spiral pattern spanning over two image slices. The first five global mergers are ISM_LL mergers, with each global vertex being pointed to by two local vertices. Figure 6 shows the GSG of the spiral image in Figure 5 starting from position P1. At that position each global vertex has two direct children. This is reflected in their child counters. At position P2, the image segment labelled with local label L15 is detected as ended. Therefore, its FV is added to its global vertex and the CC of G5 is decremented. Afterwards, at position P3, an ISM merger of image segments pointing to global vertex G4 and global vertex G5 is detected. This is reflected in an edge from G5 to G4 in the global segment graph. G5 is now a direct child of G4, so the CC of G4 has to be incremented. The next change in the global segment graph occurs at position P4, where the local image segment labelled L25 ends resulting in decrementing its parent's vertex's CC. This has the effect that now the CC of G5 is zero and the vertex feature vector is now merged into its parent. Therefore, G5 is no longer needed and can be released, the CC of its parent vertex is decremented. The following steps are described in Figure 6 until position P8 where all local image segments of the spiral image have ended. At this point, the feature vector for the complete spiral is associated with the root global vertex G1. With the end of the last image segment L11, the CC of global vertex G1 decrements to zero indicating that the global image component is finished.

3 High-throughput architecture for parallel connected components analysis

The proposed hardware architecture for connected component analysis gains its speedup through parallel pro-

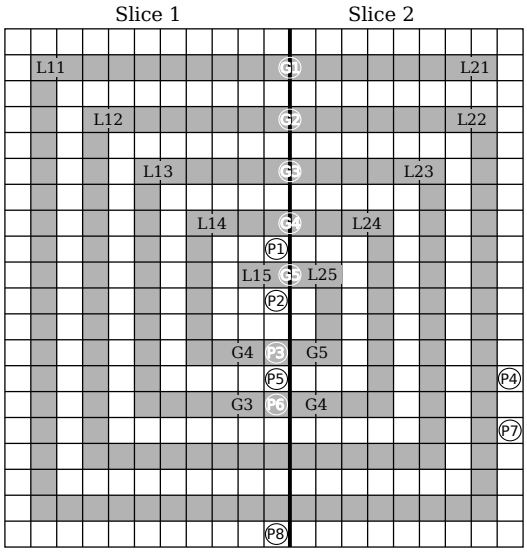

Fig. 5 Example image of spiral.

cessing of the pixel data stream. This is achieved by partitioning the received image into image slices and distributing them to several processing units. For the hardware architecture, this means that several image pixels are processed simultaneously in one clock cycle. In order to keep track of the relations between the image segments, each slice processing unit collects information about the relationship of the border pixels to the corresponding pixels of neighbouring image slices. This information is forwarded, together with the feature vector of each image segment, to the coalescing unit, where it is analysed and the feature vectors of adjacent image segments are merged to form the ultimate feature vector of the entire connected component. The proposed architecture is an advancement of the architectures described in [12] and [8]. While the architecture proposed in [12] is memory efficient, due to its scheme for reusing memory resources, and has a low latency for processing the image pixel data, it can only process up to one pixel per clock cycle in the best case. In order to achieve a higher processing bandwidth, the architecture in [8] follows the idea of processing several pixels per clock cycle and collecting information on all image segments spanning over several image slices which are evaluated at the end of the image. Although this approach offers a large processing bandwidth, a large memory is potentially required to store information from all image segments until they are evaluated at the end of the image. The evaluation of the image segments at the end of the frame also adds to the processing latency.

The proposed architecture introduces an on-the-fly coalescing scheme for image slices processed in parallel, which has two major advantages over the architectures proposed in [12] and [8]. When an image segment spanning several image slices is finished, all feature vectors of the individual image segments it consists of are already merged. Therefore, all the memory required for storing the information collected on the image segments can be freed and reused in order to store information on subsequent image segments appearing in the pixel stream. This scheme to reuse memory resources contributes significantly to the reduction in memory resources in the proposed architecture and to the reduction of processing latency which leads to high processing bandwidth, low memory requirement and low latency. Therefore, the performance of the proposed architecture can be increased significantly for the same amount of FPGA resources compared to [12] and [8].

In Figure 1 the architecture is shown as a block diagram consisting of several entities, which are described in more detail in the following paragraphs.

3.1 Slice Processing Unit

The slice processing unit (SPU) performs CCA for the pixels of its image slice. The architecture depicted in Figure 7 is an advancement on the architecture from [12]. While the general principle for processing the pixel stream is retained, major changes are necessary in order to make the immediate merging of image segments spanning several image slices possible. In order to represent the global and local vertices and their properties, the label selection needs two different label types, namely: local labels representing local image segments and global labels representing global image components consisting of several local image segments. After delaying the labels from the neighbourhood context block for one row in a row buffer, the labels reach the merger table (MT), which has the task of keeping a record of mergers among local image segments. The link table (LT) stores the connections from local labels to global labels, hence representing the edges from local vertices to global vertices in the GSG. The feature vectors associated with the local image segments are stored in the data table (DT).

A second fundamental change is the usage of an advanced scheme for memory reuse, which is a key feature for saving memory and, therefore, results in a hardware architecture requiring less resources. The scheme for reusing memory resources is efficient because one entry in the tables is required for each image segment indexed by the local label of the image segment, exploiting the fact that a maximum number of $\frac{W_{image}}{2}$ image segments have to be maintained at one time when traversing the image in raster scan manner, where W_{image} is the image width. Hence, the entries in these tables can be reused when an image segment is finished. The idea of this scheme is that, when a local label has been assigned to an image segment, the image segment keeps this label until it is finished; this is followed by the readout of the corresponding feature vector, so the memory resource can be invalidated and reused. This functionality is reflected by a changed data table and a new label

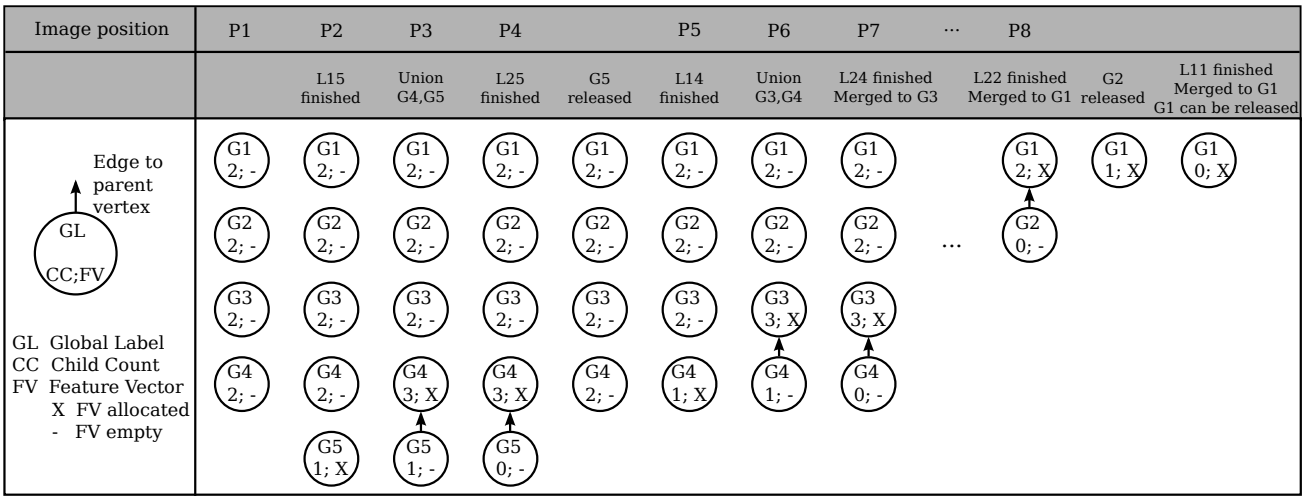

Fig. 6 Processing of the global mergers and finished image segments of the image in Figure 5.

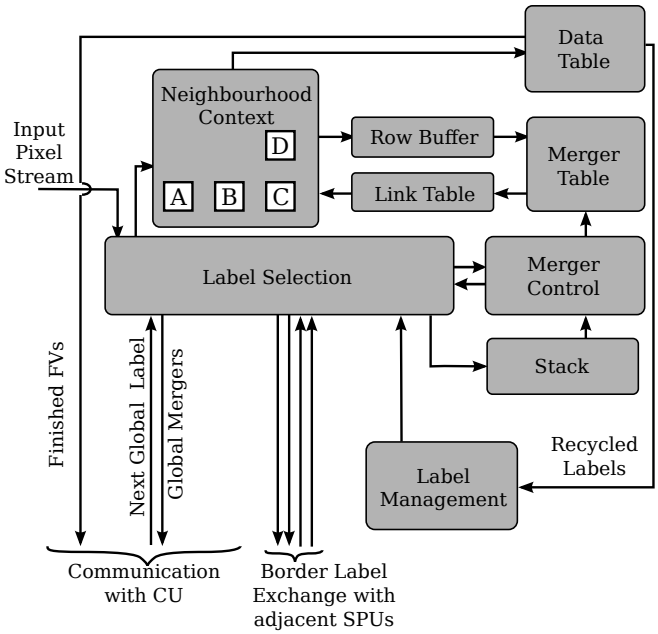

Fig. 7 Architecture of slice processing unit (SPU).

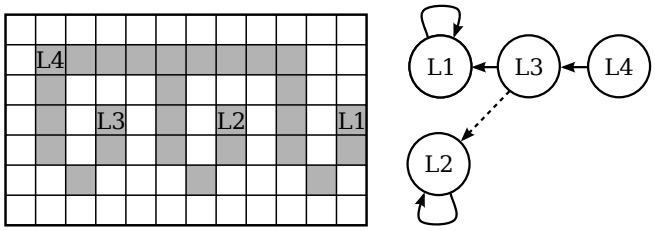

Fig. 8 Image in which augmented labels are required.

management unit unit (LM) in the slice processing unit architecture depicted in Figure 7. If the local labels in the current pixel's neighbourhood context are all background, a new label is requested from the label management unit. Due to the reuse scheme, it cannot be ensured that the labels are in numerical order. The property defined in [3] to always assign the minimum label in the neighbourhood to the current pixel can, therefore, result in wrong associations for mergers among image segments because a single look-up in the merger table does not necessarily yield the root label. This is illustrated in the example image in Figure 8. For this reason the concept of using augmented labels is introduced. The label is augmented with the row number it is generated in. The complete augmented label is interpreted for merger decisions, while only the label is used for accessing the tables. This ensures that the augmented label created earlier during processing is always smaller leading to correct behaviour [3] when a merger occurs.

In order to reuse a label two cases can apply: if an image segment is finished its local label can be reused immediately. The second case is when a merger occurs between two local labels. The invalidated label may still be in the row buffer, which would be converted to the correct label by the merger table when processing the next image row. Therefore, it is necessary to wait until the following row before reusing the label.

Each image segment has a local label which is used as an index for the memory resources within the SPU. If the image segment spans several image slices it also has a global label referencing the memory resources in the coalescing unit. Each SPU has a link table to translate from the label of the local image segment to the global label in the GSG. The fact that every image pixel has both a global and a local label is also reflected in the label selection unit, as well. The local slice labels are selected using the scheme introduced in [12] by analysing the pixel neighbourhood and choosing the minimum augmented label as a new local label for the current pixel. All global labels in the pixel neighbourhood are considered to determine which global label to assign to the current

pixel. If no global label is in this neighbourhood, the current pixel's global label is set to zero. If one label is in the neighbourhood, it is copied to the current pixel. The presence of two different global labels requires a global merger which is significantly more complex than the case of merging local labels, because the merger type as well as the labels of the pixels from the neighbouring slices have to be considered. The type of the global merger and its global labels are forwarded to the the coalescing unit by adding it to the merger queue (MQ). Additionally the SPU's link table is updated to the new global label. The long combinational path of the decision tree for determining the current pixel's global label is optimised by introducing a pipeline stage. This is allowed due to the fact that the following pixel is either of the same image segment or of background.

In order to store the feature vectors of each image segment, the data table has one entry indexed by the augmented label. Due to the label reuse principle, the labels for the image segments and hence the data table entries, are reused after an image segment is finished. The end of an image segment is recognised using the following property: when processing an image row, all data table entries of image segments which pixels are contained in the currently processed image row are updated. Therefore, a data table entry which was updated in the previous image row, and not updated in the current image row, is detected to contain a feature vector of a finished segment which can be read out. The data table entry can afterwards be reused for storing feature vectors of image segments appearing further down in the image. This behaviour is realised by adding two additional control flags to each data table entry. A valid flag is used to determine whether a data table entry is valid. Using this flag, entries are invalidated after two image segments are merged to a single one or after readout. In order to determine whether an image segment is finished, an active tag is added to a data table entry. Whenever new data is written to the data table, this tag is updated to indicate that the data table entry was updated in the current row. The tag is changed at the end of each image row. In this way data table entries in which the active flag has not been updated during the processing of the current row, contain feature vectors of finished image segments. These are detected and read out while processing the next image row by checking the active tag and valid flag of all data table entries in ascending order. If an entry is found to be valid and its active tag indicates that the last update was two rows before, it contains a finished image segment. After readout, the data table entry can immediately be reused for storing feature vectors of further image segments. For this reason, the local label to be reused is forwarded to the label management unit.

For the implementation of the data table using the Block RAM primitives available on state-of-the-art FPGAs, it is crucial to analyse the required number of read and write ports necessary. When considering a labelling

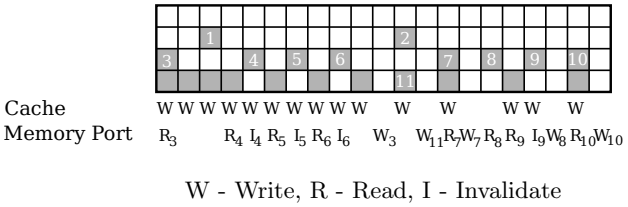

Fig. 9 Port access to data table Block-RAM.

process, several read and write operations have to be performed when processing a pixel, which has several different labels in its pixel neighbourhood. For mergers, two read operations and one write operation are required when using a naive approach, where every data table entry is read whenever it is required for processing. However, one can exploit the property that, if the currently processed image pixel is a foreground pixel, its neighbour pixel will either belong to the same image segment or be a background pixel. By caching the feature vector of the current image segment, only a single read of the data table is required for a merger, and a single write on the background pixel at the end of a sequence is required. Figure 9 shows how to cover different patterns such as single mergers or a combination of several mergers. Using this scheduling scheme, only one memory port is required to access the data table for reading and writing entries during processing [3]. The second memory port is used to read out and invalidate feature vectors of finished image segments. This makes it possible to use only a single Block RAM in true dual port mode [1], compared to the naive approach, where at least three Block RAMs are required.

3.2 Coalescing unit

The coalescing unit (CU) has the tasks of establishing the relationship between local image segments identified by the individual slice processing units (SPUs), and detecting the end of the resulting global image segments. These two tasks are executed in parallel to the pixel processing taking place in the SPUs and has the advantage that the FVs of the global image segments can be composed of the FVs of the local image segments on the fly, which enables the possibility of reusing the global memory resources after each global image segment is completed.

An SPU requires up to an addition of an extra 20% clock cycles (in average) at the end of the row for processing worst cases images [3]. The CU requires multiple clock cycles to process each ISM, SM or FO issued by the SPUs, therefore, operates on an instruction level rather than on a pixel level. Depending on the number of image slices, the number of ISMs, SMs and FOs is significantly lower than the number of pixels per image slice. Therefore, for correct on-the-fly processing, a single CU must

use fewer clock cycles to process ISMs, SMs or FOs than the SPUs require to process the pixel data of their image slices. This is analysed in detail in Section 4 and an upper bound for the number of image slices is determined.

The architecture of the CU with its sub-entities is depicted in Figure 10. The functionality and the structure of each sub-entity is described in detail in the following paragraphs.

The information collected on global mergers and FVs of finished segments are en-queued in FIFOs when entering the CU. For each connected SPU processing one image slice, two FIFOs are required: the merger queue and the finished object queue. While the merger queue stores the type of and the labels involved in the mergers, the finished object queue stores feature vectors of local image segments which the SPUs have detected to be finished. The global merger table (GMT) stores the edges between global vertices of the global segment graph (GSG). Additionally, the find operation and the path compression are integrated into the GMT, having the feature that whenever the root label of any global label is requested, the tree structure mapped to the GMT memory is collapsed, so that subsequent find operations benefit from this. The FVs of the global image segments, together with the associated child counters, are stored in the global data table (GDT). Each global label represents one entry in each of the GMT and GDT. The global label management unit (GLM) is responsible for providing global labels to the SPUs instantaneously, where each global label represents one entry in the GMT and one entry in the GDT. The GLM has one counter for providing global labels and one global label queue for storing recycled global labels, for each connected SPU. The counters are incremented in ascending order as long as the maximum number of global labels is not exceeded. Whenever a global label of a finished global image segment is passed to the GLM, it is en-queued in one of the global label queues. In order to minimise the required queue depth, the global labels on the global label queues are always prioritised. As a central controlling instance, the coalescing control unit has the responsibility of providing the communication between the different sub-entities, which includes processing of global merger and detection of finished local image segments. Mergers are processed sequentially due to limited number of memory ports of GDT and GMT and their data dependencies. The task of merger handling is hereby perceived by identification of the merger type and update of the GMT and GDT according to the description in Section 2. At the end of a global image, which is recognised by a child counter of zero in the GDT entry, the FV of the finished global image segment, is passed on to the output, so the used memory entries are no longer required and can be used for processing the next global image segment. In order to reuse its global label, it is passed to the GLM, which is now able to provide it to the SPUs for processing the next global image segment.

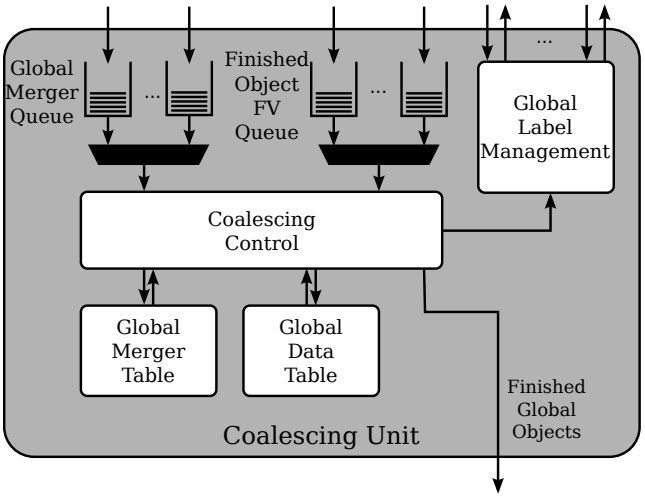

Fig. 10 Architecture of the coalescing unit.

4 Experimental results and discussion

To analyse the performance and to find out whether real-time requirements can be met, it is crucial to determine the maximum number of mergers which can occur in the worst case. The CU carries out three different types of operations: inter slice mergers (ISMs), slice mergers (SMs) and finished local image segments (FO). The number of cycles for performing the maximum possible merger operations, depends on the image width W_{image} and the number of image slices p processed in parallel. Due to this relation each image slice has a width of $W_{slice} = \frac{W_{image}}{p}$. For analysis of the maximum number of mergers, the properties of the individual operation types can be used. ISM is the only merger type which can generate global labels, SM has the ability to merge global labels and FO has the ability to invalidate global labels. Since an SM merger operation needs different global labels at most $\frac{W_{slice}}{2}$ SM mergers can be carried out in one row of an image slice. This requires at least $\frac{W_{slice}}{2}$ global labels. Therefore it is required that these global labels have been generated by ISM_LL operations in the rows before and not yet been merged to other global labels. This scenario can only occur every W_{slice} image rows, because at least $\frac{W_{slice}}{2}$ ISM_LLs are necessary to generate the required number of global labels. For this reason the worst case image in terms of processing time has the following properties:

1. Maximum number of global labels are generated.
2. As soon as $\frac{W_{slice}}{2}$ global labels are generated via ISM operations, they are merged using SM operations in the following row.
3. All local image segments end after the SM operations, which leads to FO operations in the following row.
4. The properties given in 1 through 3 repeat periodically every W_{slice} rows in the image.

Table 1 Maximum number of image slices for different image slice width.

slice width	p_{max}
96	11
128	16
160	19
192	23
224	26
256	32

An example for such a worst case image is given in Figure 12 showing the properties stated in 1 through 3 as dark pixels and the repetition as light pixels.

In order to be able to process an image stream in real-time, it is necessary to check whether the maximum number of merger operations for all processed image slices can be processed successfully within the number of clock cycles available. Due to a large tree structure in the global merger table in coalescing unit, which changes significantly with every merger and every finished image segment, an analytical exploration of the worst case tree and the number of steps for processing it for any possible image, is out of the scope of this work. Yet, to be sure that high throughput can be met, images according to the previously defined properties were generated and simulated with the CCA hardware architecture to determine the maximum number of slice processing units that can be processed by the coalescing in the worst case. The results of these simulations in Table 1 show the maximum number of image slices p_{max} which can be processed by the coalescing unit for image slices of different width.

The hardware architecture for parallel CCA introduced in Section 3 was described in VHDL (VHSIC Hardware Description Language) and implemented for *Xilinx Virtex 6 VLX240T-2* (speedgrade -2) FPGA devices.

The FPGA resources required for the extraction of the bounding box features are shown in Table 2 for different image sizes and for varying number of image slices. The diagram in Figure 11 depicts the number of Registers, LUTs and BRAMs necessary for the CCA architecture. The FPGA resources grow linearly with the number of image slices. In addition, the maximum frequency of the CCA architecture remains almost constant over the whole range of image sizes and slices. This provides good scalability with processing throughput. For the analysis of the maximum throughput, the overhead of stack processing must be considered as well, which depends on the maximum number of mergers in an image row. These can be up to $\frac{W_{slice}}{2}$ cycles for a single row, but cannot exceed $\frac{W_{slice}}{4}$ cycles in average time for the reason that there cannot be any merger in the row followed by a row having the maximum number of mergers. The throughput therefore is $T = f_{max} \times p \times 0.8$.

A comparison of the memory requirements for the proposed architecture and [8] is shown in Figure 13. The size of coalescing unit can be reduced by a factor of 42.

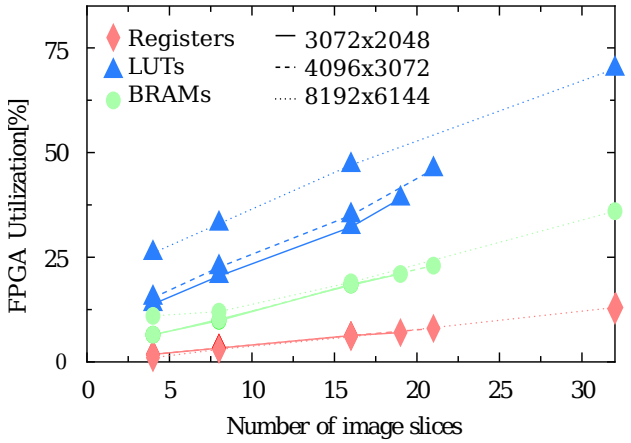

Fig. 11 Diagram of FPGA resource requirement for different image sizes.

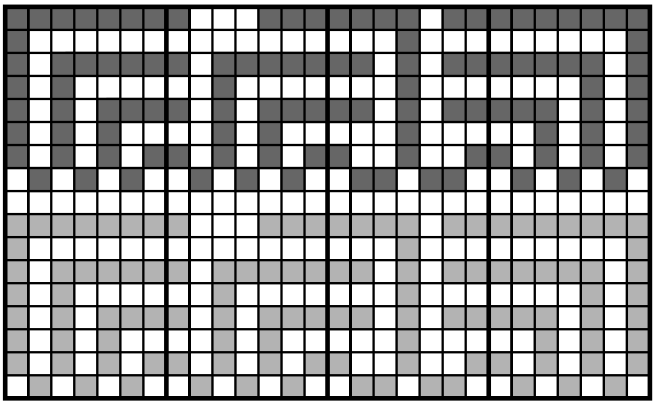

Fig. 12 Worst case image with maximum number of global mergers.

This has a huge impact on the complete architecture, so that the memory requirements for complete architecture can be reduced by a factor of more than 5. The increase in LUT requirement results from the fact that the finished object queues and merger queues, realised as distributed RAM, are included in the coalescing unit, rather than in slice processing unit in [8] and is therefore not important for a direct comparison.

Table 3 gives a comparison of the proposed architecture for connected component analysis to other hardware architectures. This shows that processing throughput is significantly higher compared to [3, 7, 12, 17, 10]. Compared to [8] the achievable throughput is similar, while the proposed architectures has a significantly reduced hardware requirement on the FPGA.

5 Conclusion

The proposed parallel connected components analysis (CCA) algorithm and architecture allow the extraction of properties of image objects with a high throughput at low FPGA resource requirements. The high throughput

Table 2 Resource requirements for parallel CCA architecture for different image sizes
Target Device: Xilinx Virtex 6 XC6VLX240T speedgrade -2

slices p_{max}	*Registers*	*LUTs*	*BRAMs*	f_{max} *[MHz]*	T *[GPixels/s]*
1024 × 512 pixels					
10	10447	25187	42	136.4	1.1
2048 × 1024 pixels					
16	17376	42792	74	137.9	1.7
3072 × 2048 pixels					
19	22452	59305	90	137.3	2.0
4096 × 3072 pixels					
21	25177	70171	99	132.6	2.2
8192 × 6144 pixels					
32	39680	106897	153	125.8	3.2
	of 301k	of 150k	of 416		

Table 3 Comparison of throughput with other hardware implementations.

	Parallelism $[\frac{Pixels}{cycle}]$	f_{max} [MHz]	Technology	Max T $[\frac{GPixels}{s}]$
[3, 7]	1	N/A	Spartan-II	N/A
[12]	1	40.63	Virtex II	0.04
[10]	6	100	Virtex 5	0.6
[11]	4	100	0.35 um	0.4
[8]	32	138.8	Virtex 6	3.5
[17]	1	95.7	Virtex II	0.1
This work	32	126.8	Virtex 6	3.2

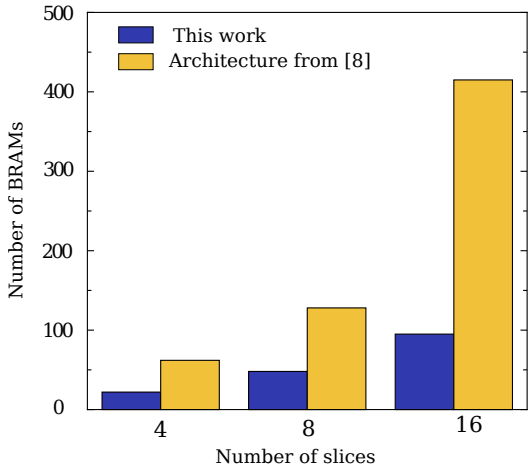

Fig. 13 Comparison the proposed architecture and [8] for image size 2048x1024.

is achieved by image partitioning results from an on-the-fly coalescing principle for parallel processed image partitions, and an advanced scheme for memory reuse, reducing memory requirements and processing latency significantly. The FPGA resources necessary for the proposed coalescing unit, which is a key component for parallel CCA, could be reduced by a factor of up to 42 compared to a previously proposed architecture. This reduces the memory requirements of the complete CCA architecture by a factor of more than 5 and enables the realised system to perform parallel connected component analysis with more than 3 GPixels per second at signicantly less FPGA resources than previous architectures.

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

References

1. (2011) Xilinx User Guide - Virtex-6 FPGA Memory Resources UG363 (v1.6). Xilinx Inc.
2. Appiah K, Hunter A, Dickinson P, Owens J (2008) A run-length based connected component algorithm for FPGA implementation. In: International Conference on Field Programmable Technology. FPT 2008, pp 177 –184
3. Bailey D, Johnston C (2007) Single pass connected components analysis. In: Proceedings of Image and Vision Computing New Zealand 2007, pp 282–287
4. Bailey D, Johnston C, Ma N (2008) Connected components analysis of streamed images. In: International Conference on Field Programmable Logic and Applications (FPL 2008), pp 679–682
5. Galler BA, Fisher MJ (1964) An improved equivalence algorithm. Communications of the ACM 7(5):301–303
6. Hopcroft J, Ullman J (1973) Set merging algorithms. SIAM Journal on Computing 2(4):294–303
7. Johnston C, Bailey D (2008) FPGA implementation of a single pass connected components algorithm. In: 4th IEEE International Symposium on Electronic Design, Test and Applications, 2008. DELTA 2008., pp 228 –231

8. Klaiber M, Rockstroh L, Wang Z, Baroud Y, Simon S (2012) A memory-efficient parallel single pass architecture for connected component labeling of streamed images. In: International Conference on Field-Programmable Technology (FPT), pp 159–165
9. Klaiber MJ, Bailey DG, Ahmed S, Baroud Y, Simon S (2013) A high-throughput FPGA architecture for parallel connected components analysis based on label reuse. In: 2013 International Conference on Field-Programmable Technology (FPT), pp 302–305
10. Kumar VS, Irick K, Maashri AA, Narayanan V (2011) A scalable bandwidth-aware architecture for connected component labeling. In: Voros N, Mukherjee A, Sklavos N, Masselos K, Huebner M (eds) VLSI 2010 Annual Symposium, Lecture Notes in Electrical Engineering, vol 105, Springer Netherlands, pp 133–149
11. Lin CY, Li SY, Tsai TH (2010) A scalable parallel hardware architecture for connected component labeling. In: 17th IEEE International Conference on Image Processing (ICIP), pp 3753 –3756
12. Ma N, Bailey D, Johnston C (2008) Optimised single pass connected components analysis. In: International Conference on Field Programmable Technology. FPT 2008, pp 185 –192
13. Rosenfeld A, Pfaltz JL (1966) Sequential operations in digital picture processing. Journal of the ACM 13:471–494
14. Seidel R, Sharir M (2005) Top-down analysis of path compression. SIAM Journal on Computing 34(3):515–525
15. Tarjan R, van Leeuwen J (1984) Worst-case analysis of set union algorithms. Journal of the ACM Volume 31 Issue 2:245 – 281
16. Trein J, Schwarzbacher AT, Hoppe B, Noffz K, Trenschel T (2007) Development of a FPGA based real-time blob analysis circuit. In: Irish Signals and Systems Conference, Derry, Northern Ireland, pp 121–126
17. Zhao F, Lu HZ, yong Zhang Z (2013) Real-time single-pass connected components analysis algorithm. EURASIP Journal on Image and Video Processing 2013:21

Michael J. Klaiber received the Diploma (Dipl.-Ing.) degree in electrical engineering and information technology from University of Stuttgart, Stuttgart, Germany, in 2011, where he is currently working toward the Ph.D. degree. From 2011 to 2016, was a Research Associate with the Department for Parallel Systems, Institute of Parallel and Distributed System, University of Stuttgart. His research interests include image processing on field-programmable gate arrays, computer engineering, and hardware architectures.

Donald G. Bailey received the B.E. (Hons.) degree in electrical engineering in 1982, and the Ph.D. degree in electrical and electronic engineering both from University of Canterbury, Christchurch, New Zealand, in 1985. He applied image analysis to the wool and paper industries within New Zealand from 1985 to 1987. From 1987 to 1989, he was a Visiting Research Engineer with University of California at Santa Barbara, CA, USA. He joined Massey University, Palmerston North, New Zealand, as the Director of the Image Analysis Unit in 1989. He is currently Professor of Imaging Systems in the School of Engineering and Advanced Technology, and the Leader of the Image and Signal Processing Research Group. One area of particular interest is the application of field-programmable gate arrays to implement real time image processing algorithms. His research interests include applications of image analysis, machine vision, and robot vision.

Sven Simon received the Diploma degree in electrical engineering from RWTH Aachen University, Aachen, Germany, in 1992 and the Ph.D. degree in electrical engineering from Technische Universität München, Munich, Germany, in 1996. He joined Siemens AG, Munich, in 1996 and Infineon Technologies AG, Munich, Germany, in 1998, focusing on hardware architectures and digital signal processing. In 1998, he became a Project Manager. In 2001, he became a Professor with Hochschule Bremen, Bremen, Germany, heading a research group for hardware architectures and sensor systems. In 2007, he became a Full Professor and the Head of the Department of Parallel Systems with the Institute of Parallel and Distributed Systems, University of Stuttgart, Stuttgart, Germany. He has authored numerous publications and a number of patents. His current research interests include parallel algorithms, hardware architectures, and sensor systems.