Emergent Algorithms for Centroid and Orientation Detection in High-Performance Embedded Cameras

Marcus Komann
marcus.komann@cs.uni-jena.de
Friedrich-Schiller-University
Jena, Germany

Alexander Kröller
a.kroeller@tu-bs.de
University of Technology
Braunschweig, Germany

Christiane Schmidt
c.schmidt@tu-bs.de
University of Technology
Braunschweig, Germany

Dietmar Fey
fey@uni-jena.de
Friedrich-Schiller-University
Jena, Germany

Sándor P. Fekete
s.fekete@tu-bs.de
University of Technology
Braunschweig, Germany

ABSTRACT

Due to increasing speed and capabilities of production machines, the need for extremely fast and robust observation, classification, and error handling is vital to industrial image processing. We present an emergent algorithmic computing scheme and a corresponding embedded massively-parallel hardware architecture for these problems. They offer the potential to turn CMOS-camera-chips into intelligent vision devices which carry out tasks without help of a central processor, only based on local interaction of agents crawling on a large field of processing elements. It also constitutes a breakthrough for understanding sensor devices as a decentralized concept, resulting in much faster computation evading communication bottlenecks of classic approaches that become an ever-growing impediment to scalability. Here, in contrast, the number of agents and the field size and thus the computable image resolution is extremely scalable and therefore promises even more benefit with future hardware development. The results are based on novel algorithmic solutions allowing processor elements to compute center points, moments, and orientation of multiple image objects in parallel, which is of central importance to e.g. robotics. We finally present the algorithm's capabilities if realized in state-of-the-art FPGAs and ASICs.

Categories and Subject Descriptors

I.4.7 [**Computing Methodologies**]: IMAGE PROCESSING AND COMPUTER VISION—*Feature Measurement*; I.5.0 [**Computing Methodologies**]: PATTERN RECOGNITION—*General*

CF'08, May 5–7, 2008, Ischia, Italy.

General Terms

Algorithms, Design, Performance

Keywords

Emergent Algorithms, Image Processing, Marching Pixels, Massively-Parallel, Object Detection

1. INTRODUCTION

Classic computing systems are usually based on a centralized computing paradigm. This statement also holds for the architecture of current smart CMOS cameras used in embedded systems for industrial image processing tasks. Such architectures are characterized by a CCD or CMOS sensor chip which captures the image in parallel. Subsequently, the information stored in the detector chip is serially read out pixel-by-pixel and possibly preprocessed by an FPGA before the actual image processing takes place with a DSP or a high speed microcontroller. Serial reading of the stored pixel information presents a source for a bottleneck which prevents from fulfilling industrial real-time requirements, i.e. reply times in the low *ms* range. For example, the position and the orientation of previously known objects given in one image have to be detected in 10 ms within two consecutive captured images in order to steer so-called robot assistants. These robots simultaneously work on a workpiece with a human worker. Another example are sorting tasks automatically carried out by fast robot gripper arms which also require sensors working as fast as possible. The same statement holds for automated diagnostic procedures, too.

A one megapixel camera with serial pixel addressing and subsequent processing would only be able to use a processing time of 10 *ns* per pixel to achieve a reply time of 10 *ms*. This is equivalent to the execution of one single clock cycle if a 100 MHz clock rate is applied to the processor of the embedded system. Much higher clock rates are not desirable in embedded systems because then required processor cooling techniques would make the embedded system larger, more expensive and would lead to unacceptable power consumption.

Hence, parallel processing techniques are required to meet these strict real-time requirements. Due to technological advancement of integrated circuit technology over the last

decade, CMOS sensor chips became the dominating standard for intelligent or smart sensor camera systems [14]. In comparison to CCD technology, using CMOS technology has the great benefit of allowing a whole System-on-Chip (SoC) to be integrated monolithically within the optical detector devices for the realization of high-performance visual microprocessors. Using CMOS, it is possible either to integrate millions of simple processing elements (PEs) as a SIMD (single instruction multiple data) processor array or a multiprocessor system consisting of a set of larger processors in the detector chip.

Due to many image processing algorithms being data-parallel, a SIMD architecture where each pixel is processed by one PE seems to be best suitable at first glance. However, considering the future growth in amount of transistors per area predicted by Moore's law, traditional fine-grained SIMD on-chip architectures will face serious obstacles. Their strict orientation on a central control unit causes performance-limiting bottlenecks when aspiring the integration of one million PEs in a future one billion transistors chip. It would lead to the implementation of countless data highways from the central unit to the PEs and vice versa in order to transfer operation code and to carry out inspection queries. Thus, such an approach is scarcely scalable to higher numbers of PEs.

This statement also holds for an on-chip implementation of a typical multiprocessor system, where each processor owns a control unit and operates on a partition of the image. If multiple objects have to be detected and if these objects are spanned across partitions, additional communication effort has to be carried out between the processors. This furthermore limits the scalability of such an architecture towards processing of higher pixel resolutions.

Consequently, we favor an architectural solution that exploits the technical possibilities offered in future smart CMOS optical detector devices differently. This processing scheme is based on Organic Computing [12] which is a relatively young research branch in which principles known from nature are adapted by technical systems in order to make these systems more robust and controllable despite their inherent complexity. The basic concepts exploited in Organic Computing systems are *self-organization* and *emergence*.

In general, emergence is commonly defined as an increase of order observable on a macro level as result of local interactions between small and primitive agents which operate on micro level [13]. In our case, in which we want to turn a future smart camera chip into an intelligent collective of smart pixel PEs, this refers to a set of virtual agents crawling on pixel level. These virtual agents are called *Marching Pixels* (MPs). They determine characteristic points only observable on the macro image level like center points and the orientation of objects. MPs achieve this only by means of local interaction and by stigmergic communication [3]. Stigmergic communication means an MP can leave information at a pixel position which is possibly evaluated later by other MPs, e.g. influencing their propagation direction.

Two things have to be accomplished in order to perform these tasks. First, the propagation of the MPs has to be worked out locally. And second, low-level arithmetic operations have to be defined to support the calculation of the objects' attributes. The controlled emergence for the propagation direction is realized by a corresponding cellular automaton (CA). The low-level operations are based on the calculation of moments using adders and very small memory units. Both is then realized in an array of smart pixels PEs.

This emergent computing scheme is more robust and scalable compared to SIMD or MIMD parallel solutions which are based on centralized control units. Failure of a single MP does not necessarily lead to a crash of the whole system like in a centralized architecture. Moreover, a higher degree of parallelism is given than in SIMD or MIMD solutions because several MPs are marching in parallel. This is in particular of advantage if multiple objects have to be detected and analyzed simultaneously. The computing performance is independent of the size of the processed image. As we will show, it only depends on the size of the largest object contained in the image.

The rest of the paper is organized as follows. Section 2 presents the idea of MPs and the corresponding architectural principle in more detail. Furthermore, the reference to related work is shown. In Section 3, the functional and the algorithmic behavior of MPs used to achieve the desired emergent behavior is described. This includes the CA for the steering of the MPs' propagation and the calculation of moments. The theoretical basis is explained and its correctness is proved. Section 4 shows essential structures of the aspired hardware implementation. The realization costs, the performance of a corresponding CMOS detector chip for a smart camera, and the achievable performance are simulated by hardware synthesis. Finally, the most important results are summarized in Section 5.

2. MARCHING PIXELS ARCHITECTURE AND RELATED WORK

2.1 Related work

As mentioned in the introduction, we intend to implement a kind of virtual swarm intelligence for visual microprocessors directly on chip level. The idea of implementing swarm behavior directly in hardware was also addressed in [6] and [11]. FPGAs have been used there to implement artificial ants or moving creatures. The difference to our approach is that in these works the emergent entities are realized as static data structures and that the states of these structures are updated by fast special-purpose hardware. That makes this solution unscalable concerning changing numbers of individuals. In contrast, our intention is to directly mimic the propagation process of the emergent entities on pixel level allowing arbitrary numbers of agents.

Using agent technology is more common in research work on software technology. To the best of our knowledge, our work is one of the first approaches to map swarm behavior for future visual microprocessors directly on hardware. Other comparable organic research on visual microprocessors pertain to cellular non-linear networks (CNNs) [10] where so-called reaction-diffusion architectures are directly implemented in optoelectronic hardware. In reaction-diffusion architectures, partial differential equations of second order are solved with analogue electronics in order to exhibit image processing tasks like e.g. edge-detection or trajectory planning [1]. But strict orientation on analogue solutions limits flexibility compared to a digital or hybrid approach. However, the realization in real hardware is quite advanced. First commercial solutions are already available [2].

MPs were introduced in [4] and [5]. Due to the key impor-

tance of center points and orientation of objects to industrial vision systems we focused on the detection of that information as task for our MPs. We have already found solutions [7] for center point detection in convex objects. In this paper, we present a new solution which allows determination of the center point in non-convex objects, too. For a better understanding, we begin with an explanation of some principles concerning the chip architecture of a MP solution.

2.2 Processor Element Array

The implementation of an emergent embedded system requires a specific hardware structure. As mentioned in the introduction, we use an array of processor elements (PEs) as core for a smart CMOS camera chip. Each of these PEs possesses a small control unit, a hardwired arithmetic-logical unit (ALU), some memory, and access to its four direct neighbors. Figure 1 shows the basic architecture of such an array and of a single PE. During the design phase, the size of the memory, the neighborhood, and the functionality of the ALU can be adjusted depending on the specific problems that have to be solved.

The MPs are implemented as data packets which are moved around this PE array. Such a data packet stores the state of the MP, which can e.g. be *march* for a moving MP, *final* for a MP in a final state, or *wait* for a blocked MP. A MP collects and contains information about the object, e.g. the number of object pixels it visited. This is important later to determine the area and the centroid of the scanned object because MPs do not have knowledge of the points they are moving to. MPs follow certain problem-specific rules in order to find points and to collect data which is of interest in a global view of the image. These rules are implemented in the control unit of the PEs. They correspond to a CA which controls the MP's propagation and decides how collected data has to be updated. Finding the right local rules is of course a challenge. Besides engineering, rules can also be found by evolution [8].

2.3 Observer Processor

As already mentioned, MPs have the task to find or at least to support the calculation of centroid and orientation of objects. Another important question is how that information is given out from the PE array after the MPs have fulfilled their task. This part of the hardware architecture is performed by a classic observer processor (OP) which is responsible for post-processing the image. This post-processing is carried out on the data received from the PE array. For example, the identification of objects out of a set of pre-known objects is performed by a comparison of the pre-known area of objects with the object's pixel number which was counted by a set of MPs. If the absolute difference is below a certain threshold, an object is detected. Another task of the OP is to communicate the result of the post-processed data to external resources, like e.g. the gripper arm of a robot or additional software.

The OP finds MPs which want to send by continuously scanning the rows of the PE array one after another (see Figure 2 for an overview). If any of the PEs in a specific row hosts a MP in final state, i.e. the MP has to output information, the observer grants this PE permission to write on its column's output bus. The OP collects all data arriving at this column's pin and meanwhile stops the continuous reading of the rows. It then outputs the data or exhibits

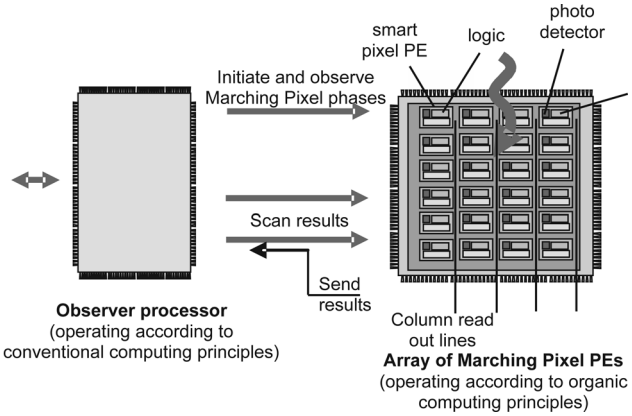

Figure 2: Architecture of the Observer Processor

further calculations. These calculations can be more sophisticated or may require facilities that are too expensive to be implemented in each PE. Of course, there should not be too many of these OP calculations because this would mean that speedup gained by exploiting the massively parallel architecture is used up by post-processing of the OP. The OP derives the position of the sending PE by combining the current row number and the specific column pin number from which it collected the data. Thus, it is not necessary to store coordinates in the PEs.

3. SYSTEM LEVEL DESCRIPTION

We need to make sure that MPs move in a valid way allowing them to collect sufficient information and to aggregate knowledge about the objects they traverse. We describe the emergent steering of the MPs here followed by the arithmetic operations MPs carry out along their march.

3.1 Flooding MPs over objects

The first task is steering several sets of MPs over the objects in an image in a way that different sets of MPs are each responsible for one object, that all pixels of the objects are visited, and that the MPs of different sets do not interfere each other. For recall, we assume that each pixel is attached to one PE which knows if it holds an object pixel or not. This distinction is the result of the process of image capturing, analogue-to-digital-conversion, and thresholding and will not be discussed here.

MPs are born at all left edge pixels, i.e. in all PEs which host pixels belonging to an object but do not have an object pixel at their directly neighbored left PE. The goal is to connect all MPs belonging to one object in a straight vertical line at the right side of the object. Thus, all starting MPs march directly to the right. Each MP continues its march as long as it finds an object pixel in front of its propagation direction. This process is repeated until it reaches a right edge pixel where it is normally blocked. MPs staying at right edges are only allowed to move on outside of the object if there is at least one object pixel somewhere on the right side of the current MP's position. In order to determine if this is true, each MP sends an information bit to the PEs directly below and above its current position. This information is stored in memory of the neighbored PEs. If a blocked MP finds such a bit in the PE in front of it, the MP continues its

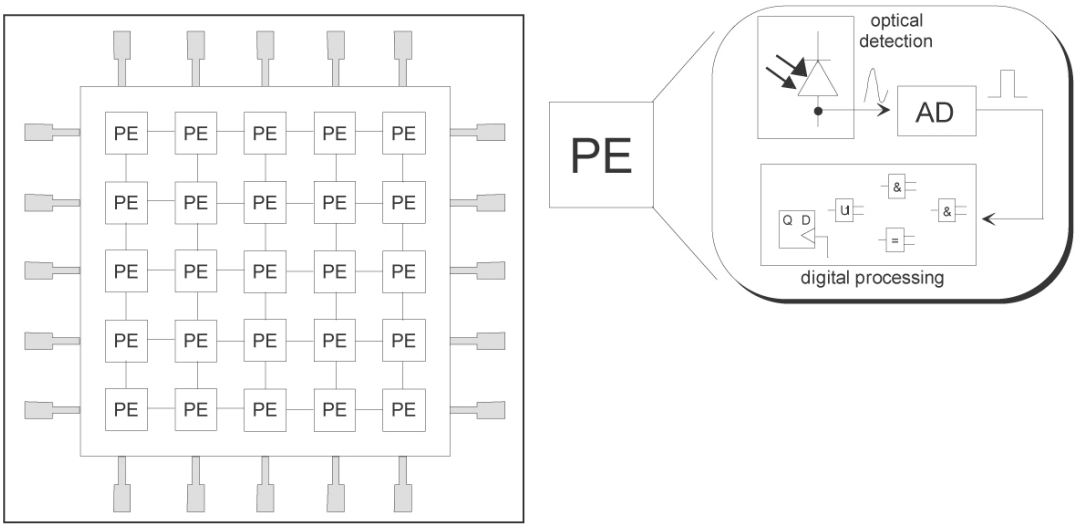

Figure 1: Architecture of a processor element array and detailed view on a single PE

march for one pixel position and also sends information to its new direct upper and lower neighbor PEs. This technique of sending bits to trigger blocked MPs starts a diffusion process that allows moving MPs over non-object pixels which are partially or completely enclosed by object pixels. Figure 3 shows the finite state machine implemented in each PE to control the MPs' march. For understanding, the information bit is referred to as *pheromone* there because it is similar to stigmergy created by pheromones although this bit is more simple and does not evaporate over time.

In the end, the straight vertical line is created at the right side of the object, i.e. at the x-coordinate of the rightmost object pixel. Each MP in this line carries information that allows determination of center points and orientation of all object pixels it crossed. Since this is solved by moving a kind of a wave of MPs from the left edge of any object to the right one, we call this idea *Flooding*. Information stored in this line is then accumulated by a MP running from topmost point of the line to lowest. How this works is topic of the next sections.

Figure 4 illustrates a sample simulation of Flooding carried out with SWARM [9]. The leftmost image shows the starting MPs at the left edges. Sets of MPs flooding as a wave over the objects can be seen in the second image, while the third image also shows MPs moving outside of objects. The final image shows the straight lines at the right sides.

Flooding supports the detection of multiple arbitrary concave objects with a relatively simple and fast algorithm. However, the price we have to pay is that the wave running over the object normally leaves the object pixels and moves outside of the object. This is a problem if different objects lie closely together. Then, different MPs might cross and thus interfere each other, making the rest of the MPs' walks wrong. However, Flooding shares this problem with other well-established procedures in image processing like the so-called BLOB analysis, in which the enclosing rectangle of objects is searched (see Figure 5 for illustration). As for an outlook, the authors are currently searching for MPs' walking schemes which don't comprise this disadvantage. Solutions seem to be possible but their requirements

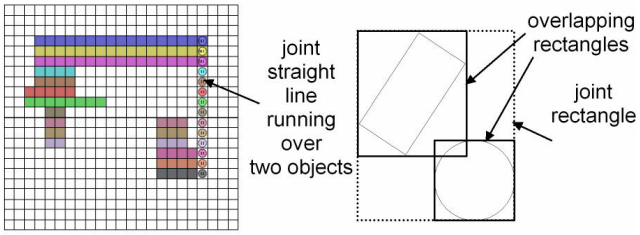

Figure 5: Problem of interfering MP swarms (left); related problem of overlapping Blobs (right)

for memory and functional capabilities of the pixel appear to be much higher. Anyhow, that is not a topic of this paper.

3.2 Moments

Moments describe attributes of the objects like centroid position or orientation. Calculating each object's moments is distributively performed during the MPs' march. To do so, an MP has to take along and update information tuples. Furthermore, it is of particular importance for the implementation that we only use basic arithmetic operations for moment calculation in the pixel PEs, i.e. addition and subtraction of limited word length. Complex operations would require too much space on a chip though.

In the following we will first give the values of interest, then show how to compute them, before finally verifying the correctness of the resulting procedure.

3.3 Problem definition

Given is an object composed of n pixels in a grid of pixel PEs. We denote the coordinates of these pixels by (x_1, y_1), (x_2, y_2), ..., (x_n, y_n). While we use these coordinates in proofs, we do not assume that these coordinates are stored in the pixel PEs. Along with an MP we pass a message with the goal of having the following information available when it stops:

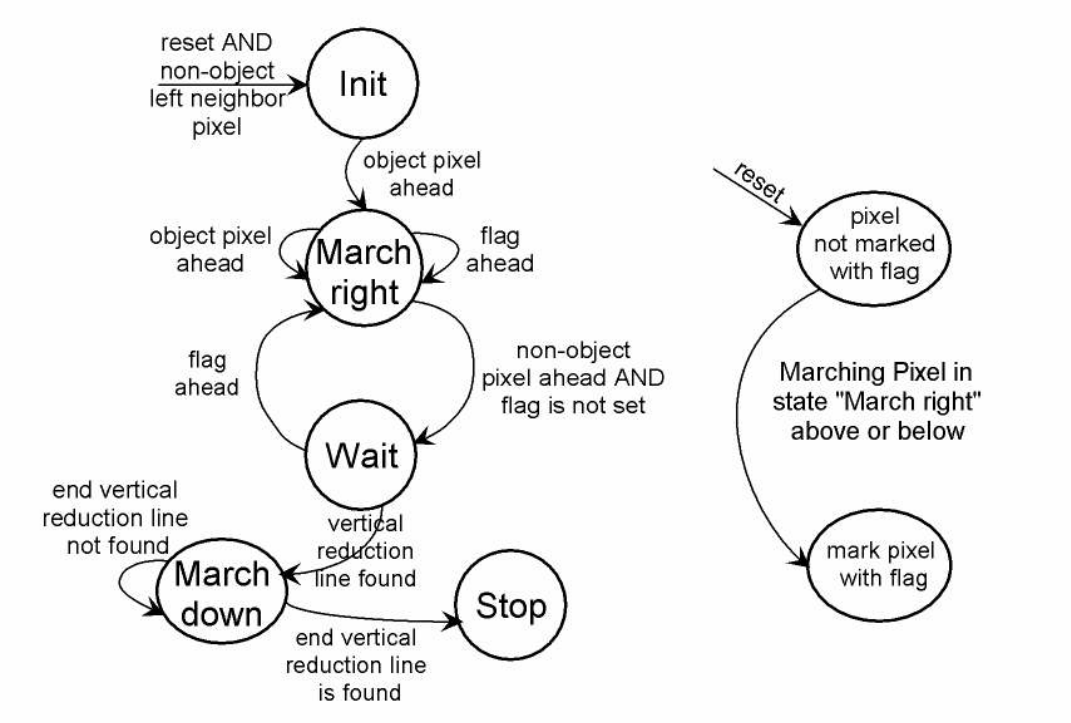

Figure 3: State machine to steer a MP's propagation and pheromone distribution

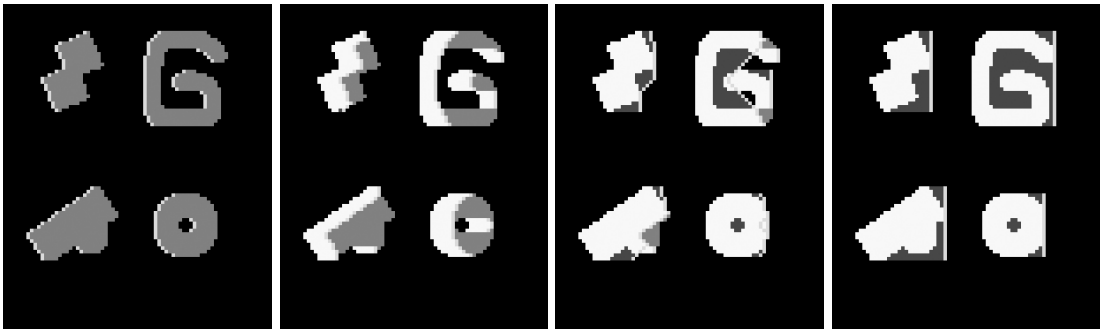

Figure 4: Simulation of Flooding

1. The center of gravity (μ_x, μ_y) of the object, i.e.

$$\mu_x = \frac{1}{n}\sum_{i=1}^{n} x_i \ , \quad \mu_y = \frac{1}{n}\sum_{i=1}^{n} y_i \ . \tag{1}$$

2. The second moments in the form of the covariance matrix

$$\begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix} \tag{2}$$

allowing to obtain the variance in every direction φ.

$$\sigma^2(\varphi) = \cos^2(\varphi)\sigma_x^2 + 2\cos(\varphi)\sin(\varphi)\sigma_{xy} + \sin^2(\varphi)\sigma_y^2 \ . \tag{3}$$

3.4 Algorithm to update relative positions

On their path, the MPs take along messages that include the relative position to the center of gravity of the object pixels they collected so far. Thus, we can obtain the center of gravity of all pixels of an object by merging the MPs that traversed it.

Therefore, we send the following 6-tuple:

$$S = (m, s_x, s_y, s_{xx}, s_{yy}, s_{xy}) \ ,$$

all of these being integers. At the beginning, each of these values is zero in every pixel apart from $m = 1$ for starting MPs.

A message sent by the pixel at (x, y) has the following meaning: m counts how many object pixels were collected by the MP so far. Assuming these were the pixels at (x_1, y_1), (x_2, y_2),...,(x_m, y_m), the other variables have the values

$$\begin{aligned} s_x &= \sum_{i=1}^{m}(x - x_i) \ , & s_y &= \sum_{i=1}^{m}(y - y_i) \ , \\ s_{xx} &= \sum_{i=1}^{m}(x - x_i)^2 \ , & s_{yy} &= \sum_{i=1}^{m}(y - y_i)^2 \ , \\ s_{xy} &= \sum_{i=1}^{m}(x - x_i)(y - y_i) \ . \end{aligned} \tag{4}$$

In Section 3.7, we show that these invariants actually hold. Consequently, a MP can easily determine the position of the center of gravity of the traversed pixels in relation to its own location: $\frac{s_x}{m}$ steps to the left and $\frac{s_y}{m}$ downwards.

The way we transmit and treat the messages has to assure that (4) is always true. Generally, all values in the tuple are sums centered at the sender. Hence, a pixel receiving the message $S' = (m', s'_x, s'_y, s'_{xx}, s'_{yy}, s'_{xy})$ will adjust the values to make itself the center. It therefore computes a new tuple S'':

- If the messages is received from the left:

$$S'' = (m', s'_x + m', s'_y, s'_{xx} + 2s'_x + m', s'_{yy}, s'_{xy} + s'_y)$$

- If the messages is received from the right:

$$S'' = (m', s'_x - m', s'_y, s'_{xx} - 2s'_x + m', s'_{yy}, s'_{xy} - s'_y)$$

- If the message was sent from below:

$$S'' = (m', s'_x, s'_y + m', s'_{xx}, s'_{yy} + 2s'_y + m', s'_{xy} + s'_x)$$

- If the message was sent from above:

$$S'' = (m', s'_x, s'_y - m', s'_{xx}, s'_{yy} - 2s'_y + m', s'_{xy} - s'_x)$$

If the wave of MPs is converging on this pixel, i.e. multiple messages for this MP arrive from different sources, they are simply added component-wise. If the pixel itself should be considered for counting in this MP, m' is increased by one.

We only use basic arithmetic operations here (addition, subtraction and left shift) that can easily be carried out in a pixel PE. For a pixel field of size $2^k \times 2^k$, we have: $m \le 2^k \times 2^k$ and need, consequently, $2k + 1$ bits for m; $|s_x| \le 2^k \sum_{i=1}^{2^k} i \le 2^{3k}$, so we will need at most $3k + 2$ bits for it (and analogous for s_y). The remaining three entries have an upper bound of $2^k \sum_{i=1}^{2^k} i^2 \le 2^{4k}$, hence, we will need at most $4k + 2$ bits.

3.5 Collecting the Results

When the MP march collapses onto a single pixel (x, y), the values of the final tuple S provide all that is necessary to compute the results described above:

1. The center of gravity (1) lies at

$$\mu_x = x - \frac{s_x}{m}, \quad \mu_y = y - \frac{s_y}{m} \tag{5}$$

This is s_x/m to the left and s_y/m downwards from that pixel, so it can send a marker MP to that position. An alternative is that an observer processor (OP) collects the values together with the position (x, y) and computes the center from the above formula.

2. The second moments are obtained by first calculating the variances

$$\sigma_x^2 = \frac{s_{xx}}{m} - \frac{s_x^2}{m^2} \text{ and } \sigma_y^2 = \frac{s_{yy}}{m} - \frac{s_y^2}{m^2} \ , \tag{6}$$

as well as the covariance

$$\sigma_{xy} = \frac{s_{xy}}{m} - \frac{s_x s_y}{m^2} \ , \tag{7}$$

giving the full covariance matrix (2). The direction of maximum variance is now described by the eigenvector for the larger eigenvalue, which is easily calculated by the OP. Now, the second moment in direction φ equals (3)

3.6 Example

For clarification, look at the example in Figure 6. After the wave has stopped, the OP collects the tuple $(5, 6, -2, 10, 2, -3)$ from the pixel at $(8, 14)$.

Using (5), the OP computes the center at $(8 - \frac{6}{5}, 14 + \frac{2}{5})$. The resulting covariance matrix is

$$\begin{pmatrix} \frac{14}{25} & -\frac{3}{25} \\ -\frac{3}{25} & \frac{6}{25} \end{pmatrix} \ ,$$

where an eigenvector for the largest eigenvalue $(\frac{3}{5})$ is $(3, -1)$, indicating the direction of maximum variance.

3.7 Correctness of the Computation

We now show that all three stages (initialization, message tuple adjustments, and result computation) are correct.

The initial values match the definition (4), so there is nothing to be shown. Assume that pixel (x, y) gets a message S' from the left, i.e. from $(x - 1, y)$, and the message conforms to (4) w.r.t. that pixel. To maintain feasibility, the

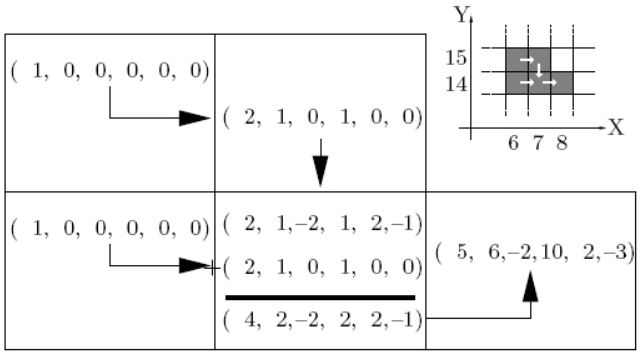

Figure 6: An example for moment-collecting MPs in a 5-pixel object.

receiver has to compute the following values:

$$
\begin{aligned}
s_{\mathrm{x}} &= \sum_{i=1}^{m'}(x - x_i) = \sum_{i=1}^{m'}(x - 1 - x_i) + \sum_{i=1}^{m'} 1 \\
&= s'_{\mathrm{x}} + m' \ , \\
s_{\mathrm{y}} &= \sum_{i=1}^{m'}(y - y_i) = s'_{\mathrm{y}} \ , \\
s_{\mathrm{xx}} &= \sum_{i=1}^{m'}(x - x_i)^2 = \sum_{i=1}^{m'}((x - 1 - x_i) + 1)^2 \\
&= \sum_{i=1}^{m'}(x - 1 - x_i)^2 \ + \ 2\sum_{i=1}^{m'}(x - 1 - x_i) \ + \ \sum_{i=1}^{m'} 1 \\
&= s'_{\mathrm{xx}} + 2s'_{\mathrm{x}} + m' \ , \\
s_{\mathrm{yy}} &= \sum_{i=1}^{m'}(y - y_i)^2 = s'_{\mathrm{yy}} \ , \\
s_{\mathrm{xy}} &= \sum_{i=1}^{m'}(x - x_i)(y - y_i) = \sum_{i=1}^{m'}(x - 1 - x_i + 1)(y - y_i) \\
&= \sum_{i=1}^{m'}(x - 1 - x_i)(y - y_i) + \sum_{i=1}^{m}(y - y_i) \\
&= s'_{\mathrm{xy}} + s'_{\mathrm{y}} \ .
\end{aligned}
$$

The other three cases (i.e. directions) are analogous and omitted here. Observe that adding pixel (x, y) itself to the calculation only increases m. Moreover, it is trivial to see that adding tuples has the desired effect.

Finally, the correctness of the resulting formulas in Section 3.5 follows directly from the invariants (4):

$$
\begin{aligned}
\mu_{\mathrm{x}} &= \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{1}{n}\sum_{i=1}^{n}(x - (x - x_i)) \\
&= \frac{1}{n}(-s_{\mathrm{x}} + nx) = x - \frac{s_{\mathrm{x}}}{n} \ , \\
\sigma_{\mathrm{x}}^2 &= \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_{\mathrm{x}})^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - x + \frac{1}{n}s_{\mathrm{x}})^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}((x_i - x)^2 + 2\frac{1}{n}(x_i - x)s_{\mathrm{x}} + \frac{s_{\mathrm{x}}^2}{n^2}) \\
&= \frac{1}{n}(s_{\mathrm{xx}} + 2\frac{1}{n}s_{\mathrm{x}}\sum_{i=1}^{n}(x_i - x) + n\frac{s_{\mathrm{x}}^2}{n^2}) \\
&= \frac{s_{\mathrm{xx}}}{n} - \frac{s_{\mathrm{x}}^2}{n^2} \ , \\
\sigma_{\mathrm{xy}} &= \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_{\mathrm{x}})(y_i - \mu_{\mathrm{y}}) \\
&= \frac{1}{n}\sum_{i=1}^{n}(x_i - x + \frac{1}{n}s_{\mathrm{x}})(y_i - y + \frac{1}{n}s_{\mathrm{y}}) \\
&= \frac{1}{n}(\sum_{i=1}^{n}(x_i - x)(y_i - y) + \frac{1}{n}s_{\mathrm{y}}\sum_{i=1}^{n}(x_i - x) \\
&+ \frac{1}{n}s_{\mathrm{x}}\sum_{i=1}^{n}(y_i - y) + n\frac{s_{\mathrm{x}}s_{\mathrm{y}}}{n^2}) \\
&= \frac{s_{\mathrm{xy}}}{n} - \frac{s_{\mathrm{x}}s_{\mathrm{y}}}{n^2} \ .
\end{aligned}
$$

The analysis for μ_{y} and σ_{y}^2 are analogous to μ_{x} resp. σ_{x}^2.

4. IMPLEMENTATION

4.1 Mapping MPs onto a parallel architecture

In order to implement this solution in real hardware, we have to map the algorithmic description of the MPs onto an appropriate parallel hardware. Starting point for this mapping process on architectural side is the parallel architecture roughly described in Figure 1. Now, the internal structure of the PE has to be worked out in detail.

A PE consists of three main parts (see figure 7): memory to store the values required for moments calculation, a control unit that implements the CA, and an ALU to carry out the necessary arithmetic modifications of the memory according to the formulas described in section 3. We need six registers as memory to store the intermediate calculations of the moments s_x,s_y,s_{xx},s_{xy},s_{yy} and the number of visited pixels m of an object. We can use either a bit-serial or a bit-parallel arithmetic for these calculations. Here, a bit-serial arithmetic is implemented in order to save chip area. In this case, the price to pay is higher computation time. It is anyway possible to carry out one million elementary steps staying below the required reply time of a few ms for an assumed clock rate of 100 MHz due to the highly parallel concept. Therefore, we can limit ourselves to the use of a full adder in each PE keeping the required area for each pixel PE low and supporting the integration of high pixel resolutions.

This also holds for the required communication paths between the PEs in the processor array. Only serial paths are required for the transfer of a MP containing the tuple. As result, we only need two unidirectional serial interconnects between PEs, one running from west to east for horizontal moving of MPs forming the vertical line, and a further one running from north to south to realize MP moving from top to bottom (see Figure 7).

Consequently, more time steps are needed to let a MP run to the rightmost/bottommost object pixel. The number of required macro steps is calculated as follows (a t'macro step consists of several microsteps, microsteps equal the clock). Let w and h be the longest extension of all objects contained in an image in horizontal and vertical direction respectively. We need $w + h$ macro steps until a MP arrives at its des-

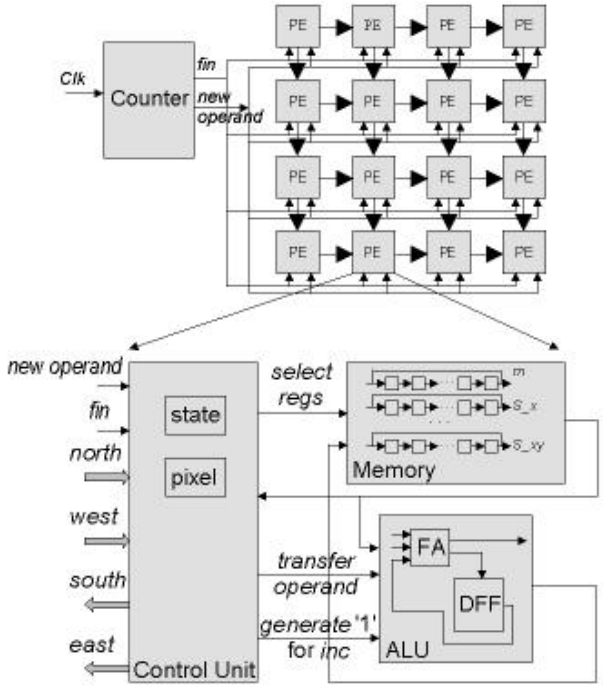

Figure 7: Block diagram of PE array and a single PE

tination point. It holds that $w + h \leq 2k$ for an image of size $2^k \times 2^k$. During one step, the transfer of all tuple components has to be finished from one PE to the next one. According to our analysis in section 3.4, we need $20 \times k + 11$ bits for an image of size $2^k \times 2^k$ for all entries of the 6-tuple to be transferred, equaling the number of micro steps. If we again assume a moderate PE clock rate of 100 MHz, one macro step will last $200 \times k + 110$ ns. In the absolute worst case, given for an object with a border line running exactly along the border line of the whole image, we would need $2k$ macro steps. Table 1 shows the required time for one macro step for different image sizes, and the time to fulfill the worst case. The worst case value is clearly below the required reply time of 10 ms. Furthermore, $w + h$ will be much smaller than $2k$ in practice.

Image size	time per macro step	worst case time
128×128	$1.5\mu s$	$21.1\mu s$
256×256	$1.7\mu s$	$27.4\mu s$
1024×1024	$2.1\mu s$	$40.2\mu s$

Table 1: Number of possible macro steps within 10 ms

Since all component values are transferred as a subsequent serial bit stream, we need a counter which announces that a macro step has finished (see signal *fin* in Figure 7. The counter uses a further signal which is becoming active when a new component value in the bit stream starts (see signal *new operand* in Figure 7). Then, the carry register in the full adder of a PE bit is cleared and a new serial addition works on another register. The registers in the memory of the PEs are organized as shift registers. The output of the shift registers are lead to the input of the full adder. The counter is not implemented in each PE in order to save chip area. This is not necessary because all PEs are working synchronously and the signals announcing the start of a new macro step or the beginning of a certain tuple value in the bit stream is the same for all PEs.

We also carried out a VHDL synthesis for Xilinx FPGAs and (our main target system) ASICs to determine which processing speed and, in particular, which image sizes are achievable now and in future. FPGAs have the advantage that we could easily use a 32bit MicroBlaze soft IP RISC processor as observer processor. For an image size of up to 32×32, the MicroBlaze can be used for post-processing determining center point and orientation of found objects.

For a smaller Spartan3-1500 device and the Virtex II VP30, the solution was carried out on an evaluation board in real hardware. Center points and orientation of two objects were found after 4286 clock ticks equaling about 21 μs for a 16×16 array using a 50 MHz clock. Post-processing in the MicroBlaze observer processor increased computing time about 50 clock cycles. A time in the μs range is a good value for position and orientation detection of multiple objects. It is far below the required reply time of 10 ms. Thus, there is enough time for further post-processing, if necessary.

Table 2 shows resource usage for different image sizes. Image sizes of 16×16 are achievable with "average" FPGAs. The clock rates meet most required response times. Anyhow, a 32×32 pixel image is already too large for all these devices. Therefore, we finally looked on synthesis results achieved for a $0.18\mu m$ CMOS process from UMC. They showed that an edge length of $2.7mm$ equaling a chip size of about $7.3mm^2$ are needed for a PE array of 32×32 including everything. That means we can integrate a resolution of 256×256 pixels with state-of-the-art technology on a chip area of approx. $4.6cm^2$. Due to the past validity of Moore's law, a mega pixel resolution seems feasible within the next five years. Timing analysis showed that a maximum clock rate of about 100 MHz is achievable. Figure 8 presents the chip layout of the final die. One can see, that the chip area is almost completely filled with logic and memory cells (light colors). Thus, the approximation of the final chip size seems feasible.

5. CONCLUSION

Building small high-performance embedded smart CMOS camera chips is a challenging task. We presented an approach which allows integrating a processing of up to one million pixels along with emergent algorithms in a complete system-on-chip which fulfills industrial real-time requirements. This is done by exploiting the possibility of using multiple local agents called *Marching Pixels* that run on a processor element array in order to find centers of gravity and orientation of multiple given objects in images.

The centerpiece of this strategy is using the calculation of moments together with a sophisticated marching strategy of MP sets. The mathematical basis was shown along with proof of correctness. Furthermore, one possible marching algorithm was introduced. The strength of this approach is its independence from image size and its ability to detect multiple arbitrary objects simultaneously. MPs are steered only by local interaction. They are running to defined characteristic pixel points within an object as result of emergent behavior. The compute time is only determined by the size

Res.	FPGA type	Logic used (%)	Slice Register used (%)	LUT (%)	f_{max} in MHz
16×16	Virtex 2 VP30	12867/13696 (93%)	5951/27392 (21%)	18786/27392 (68%)	262
	Spartan3 1500	12417/13312 (93%)	6165/26624 (23%)	18514/26624 (69%)	159
32×24	Spartan3 5000	29684/33280 (89%)	13066/66560 (19%)	44964/66560 (67%)	142

Table 2: Results of FPGA synthesis

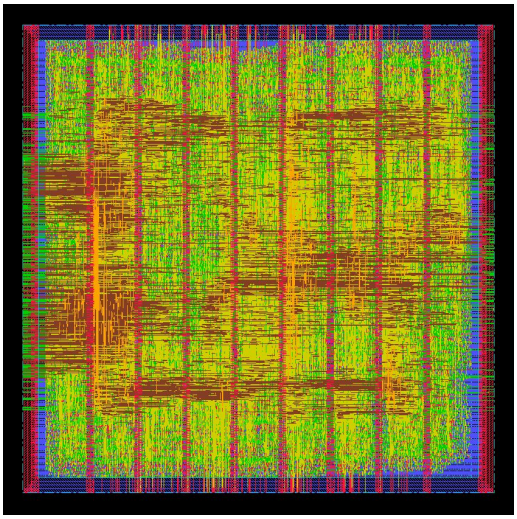

Figure 8: Resulting ASIC

of the largest object. MPs deposit collected data at characteristic points that allows determining center points and orientation.

We mapped the algorithmic scheme on a corresponding parallel hardware architecture. A worst case timing estimation showed that a bit-serial architecture is sufficient to achieve a processing frame rate of 100 Hz which is a de-facto standard in industrial vision systems. The architecture was specified in VHDL to validate the functional behavior by simulation and to carry out synthesis investigations for FPGAs and a $0.18\mu m$ CMOS process.

These synthesis results showed that a sufficient processing time can be achieved for current FPGAs but that the pixel resolution is limited below 32×32. Synthesis for an ASIC using state-of-the-art CMOS process technology resulted in an area requirement of $4.6mm^2$ for a 32×32 smart pixel cluster allowing a resolution of 256×256 on a $2.2cm \times 2.2cm$ chip. This shows that a solution based on emergent behavior of hardware agents has the potential to provide smart processing for mega pixel resolutions within the next five years.

A weakness of the presented algorithm is the need of some MPs to leave the object and the danger of interference with MPs that were created at other objects. This problem can be solved by a more complex marching scheme, where waves from opposite sides of objects steer MPs so that they don't leave the object. This strategy will be presented in future.

Acknowledgement

This paper is the product of a cooperation of computer science research groups from two German universities, namely TU Braunschweig and University Jena. Work of the authors Christiane Schmidt and Marcus Komann is funded through two different projects which are both part of the priority program 1183 *Organic Computing* of the German Research Foundation (DFG).

6. REFERENCES

[1] A. Adamatzky. Reaction-diffusion navigation robot control: From chemical to VLSI analogic processors. *IEEE Trans on Circuits and Systems*, 51(5):926–938, 2004.

[2] Gustavo Liñan Cembrano, Ángel Rodríguez-Vázquez, Servando Espejo-Meana, and Rafael Domínguez-Castro. Ace16k: A 128x128 focal plane analog processor with digital i/o. *Int. J. Neural Syst.*, 13(6):427–434, 2003.

[3] Marco Dorigo and Christian Blum. Ant colony optimization theory: a survey. *Theor. Comput. Sci.*, 344(2-3):243–278, 2005.

[4] D. Fey and D. Schmidt. Marching-pixels: a new organic computing paradigm for smart sensor processor arrays. In *Proceedings of the 2nd conference on Computing Frontiers CF'05, ACM press*, pages 1–9, 2005.

[5] D. Fey and D. Schmidt. Marching pixels: A new organic computing principle for smart cmos camera chips. In *Proc. Workshop on Self-Organization and Emergence – Organic Computing and its Neighboring Disciplines, LNI*, pages 123–130, 2005.

[6] M. Halbach, R. Hoffmann, and L. Both. Optimal 6-state algorithms for the behavior of several moving creatures. In *7th Int. Conf. on Cellular Automata for Research and Industry, ACRI 2006, LNCS 4173*, pages 571–581, 2006.

[7] M. Komann and D. Fey. Realising emergent image pre-processing tasks in cellular-automaton-alike massively parallel hardware. *Int. Journ. of Parallel, Emergent and Distributed Systems*, 22:79–89, 2007.

[8] Marcus Komann, Andreas Mainka, and Dietmar Fey. Comparison of evolving uniform, non-uniform cellular automaton, and genetic programming for centroid detection with hardware agents. In *PaCT*, volume 4671 of *Lecture Notes in Computer Science*, pages 432–441. Springer, 2007.

[9] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The swarm simulation system: A toolkit for building multi-agent simulations. 1996.

[10] Tamás Roska and Leon O. Chua. The cnn universal machine: 10 years later. *Journal of Circuits, Systems, and Computers*, 12(4):377–387, 2003.

[11] B. Scheuermann. Fpga implementation of population-based ant colony optimization. *Applied Soft Computing*, 4:303–322, 2004.

[12] H. Schmeck. Organic computing – a new vision for distributed embedded systems. In *Proc. of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005) Los Alamitos, CA, USA*, pages 201–203, 2005.

[13] T. De Wolf and T. Holvoet. Emergence versus self-organisation: Different concepts but promising when combined. *Engineering Self Organising Systems: Methodologies and Applications, LNCS*, 3464:1–15, 2005.

[14] W. Wolf, B. Ozer, and T. Lv. Smart cameras as embedded systems. *IEEE Computer*, pages 48–53, 2002.