EURASIP Journal on Applied Signal Processing 2005:17, 2788–2803

Embedded Real-Time Architecture for Level-Set-Based Active Contours

Eva Dejnožková

Centre of Mathematical Morphology, School of Mines of Paris, 35 Rue Saint Honoré, 77305 Fontainebleau Cedex, France
Email: dejnozke@cmm.ensmp.fr

Petr Dokládal

Centre of Mathematical Morphology, School of Mines of Paris, 35 Rue Saint Honoré, 77305 Fontainebleau Cedex, France
Email: dokladal@cmm.ensmp.fr

Received 14 June 2004; Revised 5 April 2005; Recommended for Publication by Luciano da F. Costa

Methods described by partial differential equations have gained a considerable interest because of undoubtful advantages such as an easy mathematical description of the underlying physics phenomena, subpixel precision, isotropy, or direct extension to higher dimensions. Though their implementation within the level set framework offers other interesting advantages, their vast industrial deployment on embedded systems is slowed down by their considerable computational effort. This paper exploits the high parallelization potential of the operators from the level set framework and proposes a scalable, asynchronous, multiprocessor platform suitable for system-on-chip solutions. We concentrate on obtaining real-time execution capabilities. The performance is evaluated on a continuous watershed and an object-tracking application based on a simple gradient-based attraction force driving the active countour. The proposed architecture can be realized on commercially available FPGAs. It is built around general-purpose processor cores, and can run code developed with usual tools.

Keywords and phrases: level set, partial differential equations, object tracking, real-time execution, embedded platforms.

1. INTRODUCTION

The level set was proposed in 1988 in [1] as a simple method to modelize or analyze the motion of a travelling interface. It offers a convenient and stable framework to implement a large variety of methods where images are seen as sets of curves. Since then, its applications have been extended to other image processing fields such as the restoration (filtering or contrast enhancement), segmentation (active contours, watershed) to the form analysis (shortest path, shape-from-shading). See [2] or textbooks [3, 4] for applications and a general overview.

From the implementational point of view, the methods can be divided into two groups: (i) filtering-like methods operating on a set of constant-level curves describing the entire image and (ii) methods that act on a single (or several) contour(s), representing one (or several) object(s) present in the image. Below, we reference these algorithms according to their computation scope, the filtering-like methods as *global-scope*-type and the active contours methods as *narrowband* type.

1.1. Scope and objectives

The objective of this paper is to open the world of hand-held, mobile devices such as PDAs, still picture or movie cameras or mobile phones to powerful image processing methods from the level set framework. The novelty of this paper resides in the presentation of a reusable architecture capable to run optimally various algorithm types from the level set family. This architecture corresponds well to the system-on-chip concept, and verifies the needs of hand-held devices concerning their energy and implementational limitations. We particularly concentrate, among other aspects, on the execution on multiprocessor, parallel, scalable architectures which is an important aspect permitting to reduce the energic consumption.

The rest of this paper is organized as follow. After reviewing the state of the art of existing implementations and accelleration attempts, analyzing the family of the level-set-based algorithms (Section 2), we present the architecture (in Section 3) that best verifies the algorithmic needs and remains efficient with respect to the HW implementation issues listed above. Its efficiency is demonstrated on a contour-tracking algorithm proposed in Section 4.2. The text concludes by presenting some benchmark results and general conclusions.

1.2. State of the art and technological difficulties

The implicit representation of the travelling interface by using the level set increases the computational effort by one order of magnitude. A faster implementation obtained by narrowbanding the computations around the travelling interface (originally called the *tube method*) was proposed by Adalsteinsson *et al.* [5] and Malladi *et al.* [6]. Despite the narrowbanding which reduces considerably the number of points to process, the level set methods remain computationally expensive, because of (i) using nonlinear functions, and (ii) a high number of iterations. The computational complexity has unpleasant consequences on both the execution time and the power consumption. If the execution time can be reduced by parallel execution (provided that the algorithm is parallelizable), the overall energy budget (following from the number of necessary operations multiplied by the energy to perform one basic operation) remains constant.

The numerous attempts to speed up the implementation of PDE-based methods made in the past were done in various axes.

(i) Algorithmic: as, for example, the implementation alternative to the level set, using spline-based modeling of the contours. Precioso and Barlaud [7] have obtained a fast execution on Pentium-based machines with spline-based active contours. A special care must be done to handle the topology changes. Cserey et al. study in [8] the implementation of linear and nonlinear diffusions on neural networks. In general, the algorithmic modifications are often applicable to only one type of algorithms.

(ii) Mathematical: proposing a faster convergence either in another space, or using another integration scheme. Weickert et al. propose in [9] the semi-implicit integration scheme, and the AOS scheme with arbitrarily large integration step for filters which can be written in a specific form as in [10]. The semi-implicit scheme increases the integration speed without affecting the numerical stability; it deteriorates only the numerical accuracy. Later, Goldenberg *et al.* [11] and Smereka [12] use a semi-implicit scheme for the active contours. However, the mathematical modifications are often applicable to only a restricted family of algorithms.

(iii) Hardware-based implementations are of three types.

  (a) *Supercomputers*: Holmgren and Wallin [13] use a self-optimizing nonuniform memory access (NUMA) supercomputer implementing a high-accuracy solver for several integration kernels. Sethian [14] has studied study flame propagation models on a CM-2 machine with 65K processors. The author reports a true massively parallel calculation with one processor per grid node.

  (b) *Graphic hardware*: Rumpf and Strzodka benefit from a high memory bandwidth and implement a nonlinear diffusion [15], and a level set segmentation [16] on a graphic card. Cates *et al.* [17] implement an active-contours-based segmentation tool on a graphic hardware to increase the interactivity when a number of parameters must be tuned to obtain a correct segmentation results. Sigg *et al.* implement a signed-distance function transform on a graphic hardware [18].

  (c) *Specific HW accelerators*: Hwang *et al.* [19] propose an orthogonal architecture designed for numerical solution of PDEs, not inevitably related to the image processing. It is built around n processing units and n^2 memory blocks. Each processor is connected to the memories by buses dedicated to only one processor, equipped with a memory access controller. The drawback of this design is that the number of interconnexions and buses increases with the square of the number of processors.

  Gijbels *et al.* [20] propose a VLSI architecture for nonlinear diffusion conceived for image improvement on image sequences. The authors use an SIMD[1] architecture with distributed memory for parallel nonlinear diffusion (i.e., the global-scope-type) used in some vision application. The estimated performances are some 100 iterations on a 256×256 image every 0.25 seconds, whereas the processing units themselves are clocked at 20 MHz.

This paper focuses on the HW-based implementation issues of the level set techniques on embedded, *one-chip devices* that will be easily (i) *scalable*, to adapt their computational power to the requirements of the chosen application, (ii) *programmable* with conventional programming tools, (iii) by far *less energy consuming* than Pentium-based desktop machines with comparable computational power, and (iv) as *small sized* as possible. The surface occupation is important because it has a direct impact on the price of both the chip itself and the embedding system (such as personal vehicles, hand-held devices, etc.). These contraints exclude both the graphic hardware and supercomputer implementations, since they do not match the objectives of one-chip devices, as well as the SIMD architecture, presented in [20], which cannot be used either because of its considerable number of used processing units (one unit per image column).

Generally speaking, it is essentially due to the algorithmic complexity that no embedded platforms have so far been proposed for the narrowband-type algorithms. The issues to handle include the following.

(i) *Nonlinear computations* employed in the integration step, *numerous iterations* necessary to obtain the convergence, and often required *floating-point accuracy* impose using fast ALUs. Their considerable surface occupation and energy consumption exclude their replication in a great number on one chip.

(ii) The *distance function* computation represents another difficulty of parallelization of the narrowband applications. Dejnožková and Dokládal [21] present a detailed analysis of existing algorithms (namely fast march-

[1] Single instruction multiple data.

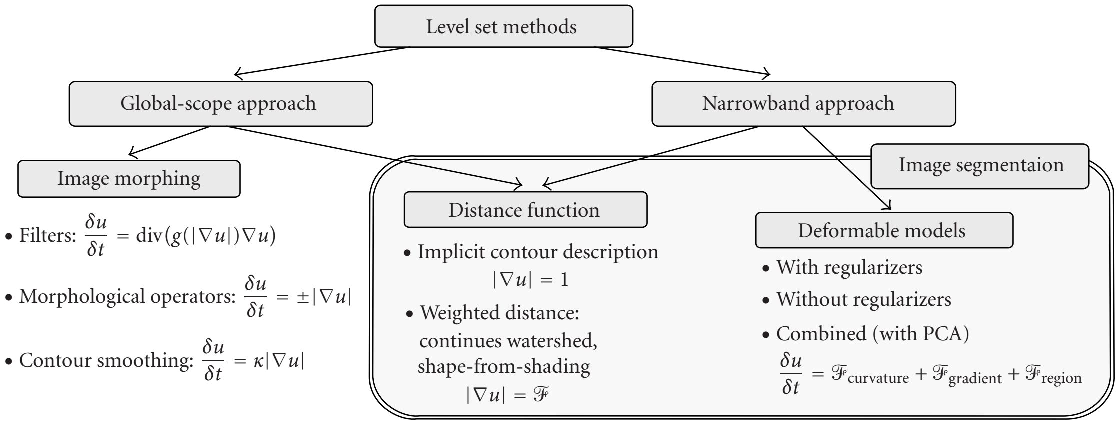

Level set methods
Global-scope approach
Narrowband approach
Image morphing
• Filters: $\frac{\delta u}{\delta t} = \mathrm{div}(g(|\nabla u|)\nabla u)$
• Morphological operators: $\frac{\delta u}{\delta t} = \pm|\nabla u|$
• Contour smoothing: $\frac{\delta u}{\delta t} = \kappa|\nabla u|$
Image segmentaion
Distance function
• Implicit contour description $|\nabla u| = 1$
• Weighted distance: continues watershed, shape-from-shading $|\nabla u| = \mathcal{F}$
Deformable models
• With regularizers
• Without regularizers
• Combined (with PCA) $\frac{\delta u}{\delta t} = \mathcal{F}_{\text{curvature}} + \mathcal{F}_{\text{gradient}} + \mathcal{F}_{\text{region}}$

Figure 1: PDE-based algorithms overview.

ing). It shows that they are sequential and ordered (explained below). The authors propose to remove the bottleneck with the introduction of massive marching [21]. It is a fully parallel algorithm, making use of a nonequidistant propagation front.

Since we aim the entire algorithm family, whose common denominator is the level set implementation, the architecture has to be maximally flexible and scalable, and maximally using the occupied silicon surface. The recently emerging dynamic reconfiguration represents an alternative solution to the tradeoff between the functional flexibility of complex systems and the occupied surface and energy consumption, see for example [22, 23]. In some cases, the advantages of the dynamic reconfiguration may however be outmatched by the drawbacks that constitute a lenghty and difficult design, the need for special design tools [24], and external circuits controlling the chip reconfiguration.

Our study demonstrates that for the level set domain, the satisfying tradeoff between the flexibility and size can also be obtained by the programmability, offered by on-chip embedded processor cores and some DSP functions.

The following section presents the analysis of the architectural choices, including the computational resources, memory consistency model, and communication management. The resulting system has been synthesized for commercially available FPGAs.[2] Their performance becomes almost comparable to the ASICs.[3] Though the ASICs still outperform the FPGAs in the energy consumption (a key feature in mobile devices), the FPGAs remain a useful prototyping platform, and a possible intermediate development step towards an ASIC.

2. ALGORITHM ANALYSIS

This section discusses hardware implementation issues of several algorithm types from the level set context. All the types consist of two basic steps: an *initialization* step that differs according to the method used, and the *evolution* step, which makes the contour(s) travel in space and/or time according to the given partial differential equation (PDE). Usually it makes use of some local integration kernel, and is repeated until stability. In general, only the use of a local information is easily parallelizable. If the image is considered as a continuous signal, then the PDEs can be seen as an iteration of a local filter operating on the neighborhood [4].

Typically, the evolution proceeds by deforming one or several curves (propagation front) or surface with a given PDE. The PDEs methods can be classified into the following categories (cf. Figure 1).

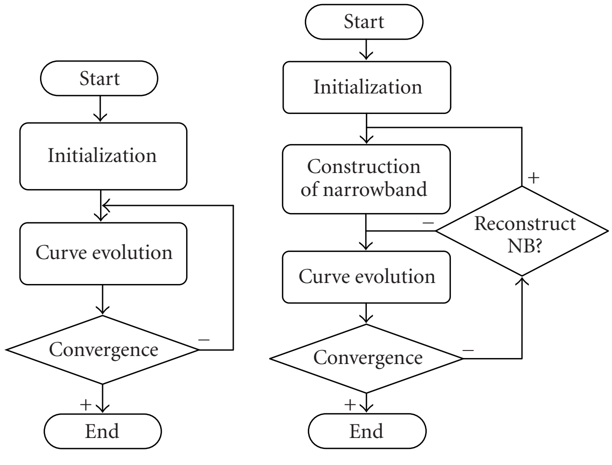
(1) Surface propagation includes *diffusion filters* [25], [26], or [27] for a more comprehensive survey, *geometric smoothing* [10, 28, 29], *denoising*, and *morphological operators* [30], [31], [32] or [33] characterized by the evolution equation $\partial u/\partial t = \mathcal{F}(u)|\nabla u|$, where u represents the evolving image. The input image represents the initial conditions u_0. All points in the image are processed in every iteration. The temporal evolution is based on the local neighborhood and generates the evolution of the level sets in the space [4]. The evolution stops as soon the convergence or the given iteration number is reached.

(2) Wave propagation includes algorithms of *weighted distance*, *continuous watershed* [34], *Voronoi tesselations* [35], or *shape-from-shading* [36] that are controlled by the Eikonal equation $|\nabla u| = \mathcal{F}$. This steady-state solution is propagated from the given sources (that may be obtained from the initial image by other means) on the entire image according to the defined speed $\mathcal{F}$. The algorithm operates locally, only on the narrowband of the evolving front. The solution is propagated in waves equidistant to the sources by using ordered data structures. This technique is being referred to as marching methods, proposed by Sethian [37], as a special case of the Dijkstra shortest-path algorithm.

(3) Deformable models. An important breakthrough in the deformable models represents the introduction of

[2] Field-programmable gate array.

[3] Application-specific integrated circuit.



(a) (b)

Figure 2: Different stages of the level set family algorithms. (a) *Global-scope*-type algorithms. (b) *Narrowband*-type algorithms.

active contours (or snakes) proposed by Kass *et al.* [38] in 1987, and deformable surfaces by Terzopoulos *et al.* [39] one year later. Another early example of deformable models represents the ballons by Cohen [40]. Implicit representation of the interface as a constant level set of another function was studied simultaneously and independently in 1993 by Caselles *et al.* [41] and Malladi *et al.* [42], and later by Malladi *et al.* [43, 44, 45]. The geodesic active contours were proposed meanwhile in [46, 47]. Another model was proposed later in [48].

We distinguish the types *with regularizers* (controlled by statistical information of regions), *without regularizers* [27], or *combined* with other techniques (e.g., principal component analysis [49]). The evolution equation writes in the form $\partial u/\partial t = \mathcal{F}_{curvature}(u) + \mathcal{F}_{grad}(u) + \mathcal{F}_{region}(u)$. The algorithms proceed by deforming a given initial contour (given by $u_0 = 0$). The deformation is controlled by internal and external forces obtained at each iteration from (i) the contour itself and (ii) the geometrical (curvature, gradient) or statistical characteristics (mean value of the region intensity) found in the image [4].

(4) Optical flow is controlled by the equations $\partial u/\partial t = f(\nabla u, I_1) + g((\partial I_2/\partial x), h)$, $\partial v/\partial t = f(\nabla v, I_1) + g((\partial I_2/\partial y), h)$. The motion vector is obtained by solving some system of the above-given equations at each point in the image (I_1, I_2 are the successive sequence images, h is the searched motion vector field) [50].

Since the nature of the optical flow algorithms differs from the temporal curve evolution principle of the three first groups, the proposed architecture does not address this type of algorithms. On the other hand, the optical flow often serves as a support for the three other types.

All the computation steps of the first three categories can be unified in two following iteration types, see Figure 2. (1) *Global-scope iteration type* includes the surface evolution. It operates *sequentially* on the entire image. (2) *Narrowband iteration type* includes the wave propagation and deformable models (curve evolution).

Indeed, applying narrowbanding to the curve evolution algorithms changes the computational aspects. The points to recalculate in every iteration are now taken from some subset of the image. This set is commonly called narrowband, and contains points situated closely (up to some chosen distance) to the current position of the travelling interface. Two types of operations are commonly applied on the narrowband: (i) the curve motion scheme itself, and (ii) the (re-)construction of the narrowband. The (re-)construction differs substantially from the other algorithm types. Indeed, all HW implementations of the active contours, cited in Section 1, use fast marching; a progressive, equidistant construction of the distance function. Fast marching itself belongs to the *wave propagation* algorithm group. It requires ordered data structures based on the priority of points [3]. From the algorithmical point of view, the ordering introduces a great data dependency, reducing the parallelization potential. From the HW implementation point of view, algorithmic ordering of the points to process introduces *random* accessing to the memory.

Parallelize the wave propagation is a tough issue, calling attention of many researches for a long time, compare a survey by Roerdink and Meijster in [51]. The recent introduction of massive marching opens the possibility to parallelize also the computation of the distance function (cf. [52] or [21]). Massive marching is similar to the fast marching method (cf. [53] or [54]) and uses the same entropy-satisfying upwind scheme. It differs from fast marching by the fact that it eliminates its sorted propagation of the solution and makes the implementation fully parallelizable, with a small grain and low data dependency.

For completeness, we mention another parallelization strategy, called group marching, developed by Kim in [55]. Group marching identifies on the front groups of points that are processed parallely in the same time. It requires nonetheless to maintain a global variable making a truly parallel implementation difficult.

The next section analyzes the execution of the different algorithm steps by considering the use of massive marching for the narrowband construction. Note that the curve evolution (both algorithm types) as well as the narrowband construction (narrowband type) are time-critical. Many iterations may be needed to obtain the convergence and the narrowband has to be reconstructed repetitively during the evolution to preserve the required properties of the implicit curve description.

2.1. Data-flow analysis of different algorithm steps

In the following, we assume that, except the methods where regularizers[4] are used, the new values that the points re-

[4] Statistical information, like colour for example, represents global variables.

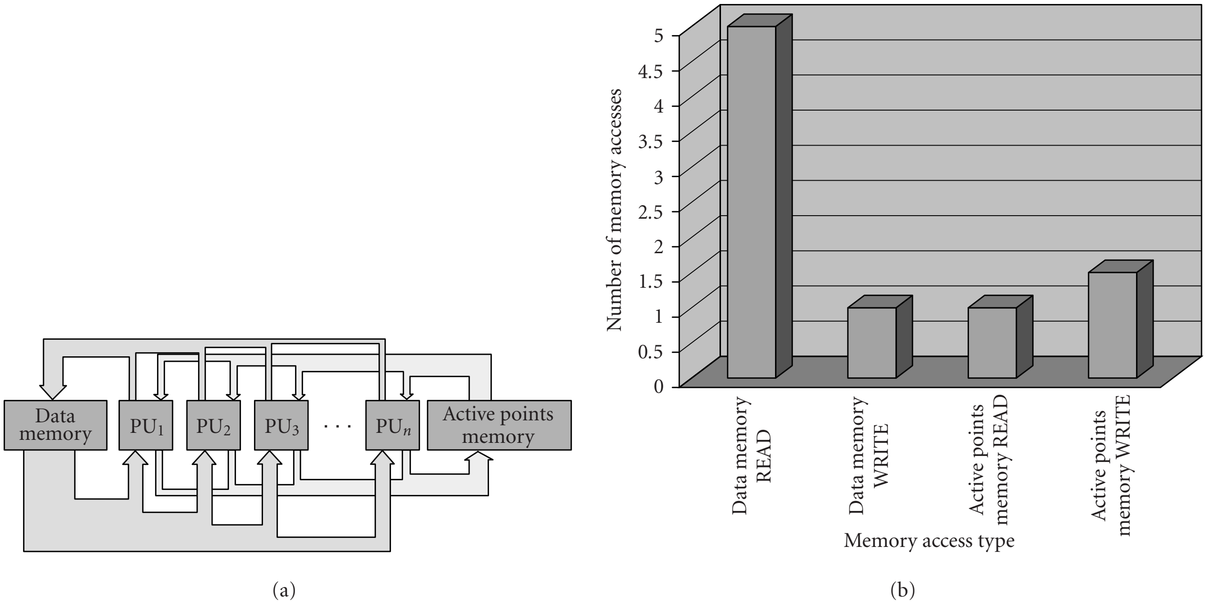


Figure 3: Curve evolution implemented by using several processing units operating in parallel: (a) data-flow chart, and (b) corresponding memory accesses.

ceive are results of operations only on local neighborhood.

Limiting the calculations to a narrowband around the travelling contour corresponds formally to operating on sparse matrices. Practically, in order to obtain the correct evolution, all active points need to be recalculated in one iteration, before the next iteration starts. Therefore, unless one uses one processing unit per point, the values u^{n+1} need to be stored separately from u^n, until all the points are updated. If this condition is verified, then the processing order is not important, and both the iteration types can be unified under the following form. The set $\mathcal{A}$, respectively, represents either the entire image or the narrowband set:

```
for all $p_i \in \mathcal{A}$ do (in parallel)
{   Retrieve_Neighborhood $u^n(N(p_i))$ and $u^n(p_i)$;
    Calculate_Value $u^{n+1}(p_i)$;
    Update_Value $u^{n+1}(p_i)$;
    Activate_New_Points (insertion in $\mathcal{A}$);      }
```

Since our constraints exclude the massive parallelism (for production cost's reasons), we adopt a semiparallel approach instead. The data-flow chart corresponding to a semiparallel execution of this code on several processing units is given by Figure 3a. The data u are stored in the data memory block (two pages for u^n and u^{n+1}). The active points memory block stores the set $\mathcal{A}$, that is, the coordinates of the points to process (not used for the global scope-type algorithms). The active points are read and processed by several independently operating processing units. Both the memory blocks are organized in two pages, for the present one and the next iteration.

The width of the paths corresponds to the volumes of transferred data. The most intensive data traffic is on the shared blocks. The READ data memory flow is five times larger than the WRITE data memory flow because the complete four-neighborhood is read to update the central value (cf. Figure 3b). Similarly, since one processed point may activate several of its neighbors, the mean WRITE active points memory flow is slightly higher than READ active points memory.

The narrowbanding of active contours techniques impose random memory access to the data memory block. These aspects will be taken into account in Section 3.

2.2. Timing analysis

To optimize the data flow, limit simultaneous accesses to the shared blocks, and obtain a balanced activity of all the used blocks, it is necessary to consider also the timing of the algorithm execution.

The global-scope type operates on the entire image, that is, each point in the image is active and the set $\mathcal{A} = \text{supp}(I)$, $I =$ image. The narrowband type operates on $\mathcal{A} = \{p \mid |\text{dist}(p)| < NB_{\text{width}}/2\}$, where NB_{width} is the width of the narrowband around the contour. For massive marching, the definition of $\mathcal{A}$ slightly differs (see [21]).

This code has two major features.

(1) The retrieval of the point's and its neighbors' values $u^n(p_i)$ and $u^n(N_4(p_i))$ requires five memory readings and is usually faster than the following calculation of $u^{n+1}(p_i)$, which usually involves nonlinear functions. During the calculation of $u^{n+1}(p_i)$, the memory block is idle.

(2) The execution of same parts of the code can have different length due to *IF*-conditions and various input values.

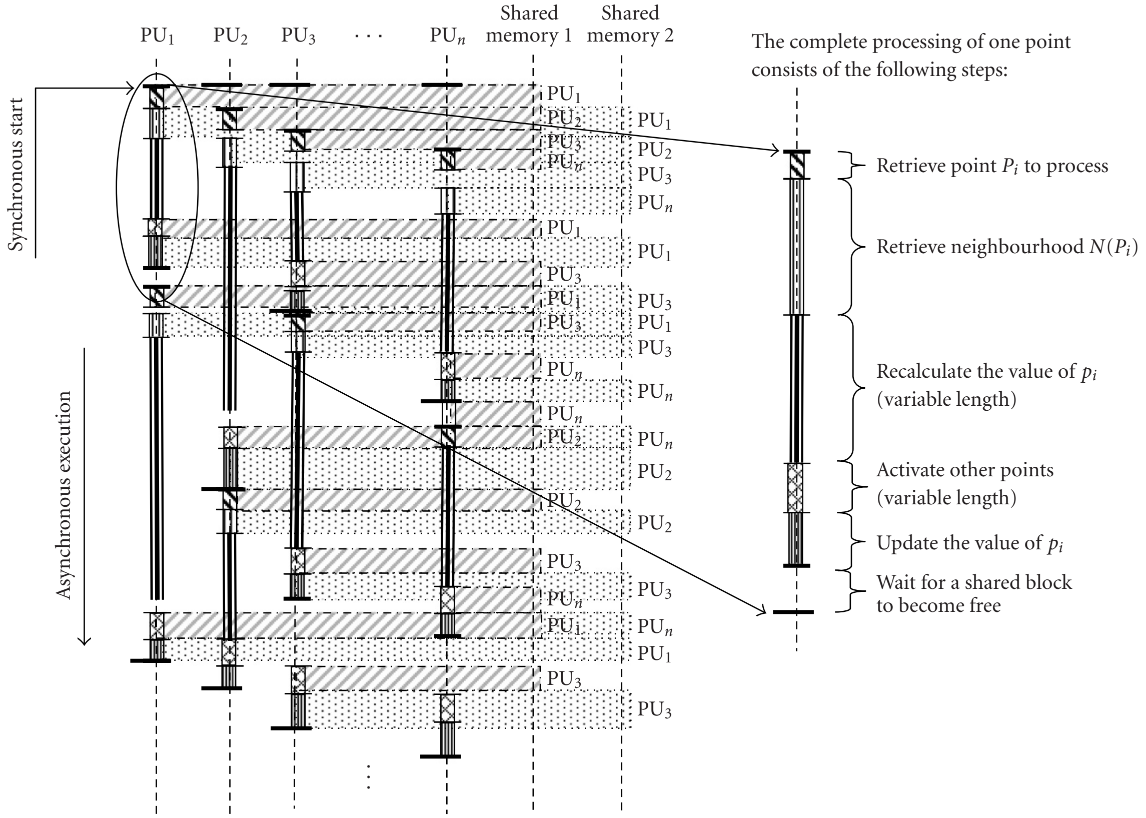

PU$_1$
PU$_2$
PU$_3$
· · ·
PU$_n$
Shared memory 1
Shared memory 2
Synchronous start
Asynchronous execution
The complete processing of one point consists of the following steps:
Retrieve point P_i to process
Retrieve neighbourhood $N(P_i)$
Recalculate the value of p_i (variable length)
Activate other points (variable length)
Update the value of p_i
Wait for a shared block to become free

FIGURE 4: Asynchronous execution of the code on four processing units (PU$_1$ to PU$_4$) and accesses (in grey) to a shared block.

Whenever the memory is idle, it can (and should) be used to retrieve other data to process. The fact that the algorithms operate locally (using only the information from the neighborhood) characterizes this algorithm family by a fine granularity. On the other hand, the nonlinear functions categorize these algorithms rather into the medium granularity group because, in most cases, a fully functional ALU is necessary to implement the computation. These considerations impose the choice of an MIMD architecture. The processors operate in the SPMD[5] mode, corresponding best to the data flow diagram given by Figure 3.

After a synchronous start of all the processing units, the variable length of some portions of the code gives birth to an asynchronous execution, (cf. Figure 4). The asynchronous execution is advantageous for parallelizable algorithms with numerous *IF*-conditions, because it randomizes the access to the shared blocks. Simultaneous accesses become rare and their HW management is easier. After the analysis of various algorithms, it becomes clear that the choice of asynchronous execution of the code on several PUs is a natural choice for the level family.

Note, that despite the asynchronous execution, the PDE-based algorithms have one or more synchronization points: the end of one iteration. This is indicated by either (i) emptiness of one of the active points memory pages (narrowband-type algorithms), or (ii) end of the raster scan of the image (global scope-type algorithms). The end of the algorithm is indicated by either (i) emptiness of both active points memory pages (for the narrowband-type algorithms), or (ii) the number of necessary iterations (both algorithm types), or (iii) the convergence (both algorithm types).

3. ARCHITECTURE

The image processing domain is known for various algorithm granularity and data dependency. Indeed, the data dependency and granularity are two factors that have major influence on the choice of parallel implementations. Historically, the fundamental model of parallel architectures has been introduced by Flynn [56]. The further effort has been concentrated, besides the computation resources, on the efficiency of the communication configurations (see Cypher and Sanz [57]).

The massive parallelism is efficient for regular algorithms with fine granularity (cf. Gibbons and Rytter [58], Broggi *et al.* [59] or artificial retine by Manzanera [60]). On the other hand, if it used for random memory access implementations, the chip activity versus occupied surface will become poor. The same arguments are valid in the case of SIMD-type architectures (see Cypher and Sanz [57] or e.g., survey in [61]).

[5] Single program multiple data—the processors execute asynchronously the same program.

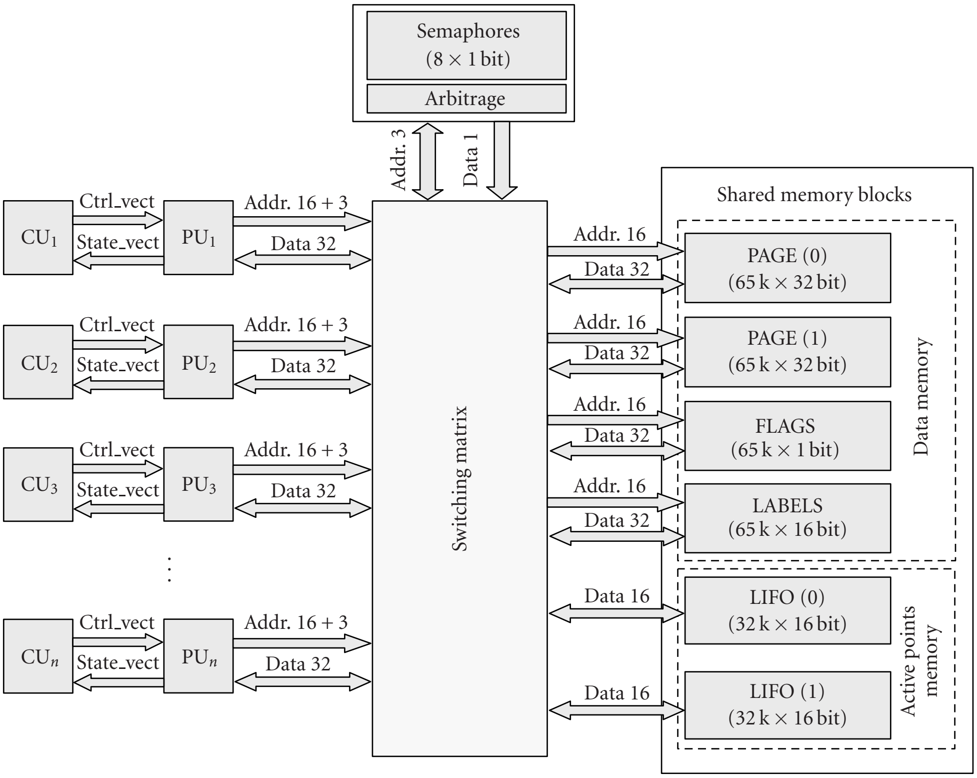


Figure 5: Global overview of the architecture.

Hence, the image analysis community started to consider, as a possible execution platform for mean and high granularity algorithms, in the late 1980s, programmable, multiprocessor, one-chip architectures. See for example [62], [63] or the survey of multiprocessor architectures with shared and distributed memory [64]. For another example and additional references, see a motion estimation on a set of video signal processors by De Greef *et al.* [65], or watershed segmentation in Moga *et al.* [66], Noguet [67], or Bieniek [68].

The data flow of the active contours, analyzed in the previous section, makes them correspond better to "weaker" parallelism models where the design effort concentrates on the task and data dependency decomposition, task scheduling, and efficient management of accessing to shared resources. The architecture template, presented in this section by Figure 5, is derived from the data-flow analysis given by Figure 3. In the following, we detail the description of the individual blocks.

Processing and control units

The computation of the propagation speed $\mathcal{F}$, which is generally a nonlinear function, is a challenge for an efficient implementation. It seems necessary to use a fully functional arithmetic logic unit (ALU).

The processing units were realized in VHDL and HandelC as a model of a RISC processor. They are equipped with a set of registers. The used data word width is 32 bits to store a fixed-point data (24 + 8 the integer and fractional part).

Every processing unit is controlled by a control unit (CU). The execution of the algorithm was simulated by coding the algorithm in HandelC. The advantage of this approach is that the functional model can be replaced by another processor model or by an embedded core available on some FPGAs.

Switching matrix

The medium granularity combined with intensive random accesses to the data memory shows that no optimum *fixed* interconnection network can be found for the level set algorithm family. Rather than using a fixed network, one can use a switching matrix which, coupled with semaphores and arbitrage, permits to any processing unit access to any shared block, provided that it is not currently being used by another processing unit. Several PUs can access simultaneously to different shared blocks.

The address buses are 16 + 3 bits (16 bits for the data addressing and 3 bits to address eight semaphores for the following eight shared blocks), four data memory blocks (data PAGE(0), PAGE(1), FLAGS, and LABELS). The active points memory is divided into two blocks, each equipped with a bidirectional reading/writing channel since each of the stacks is in one iteration either read or written but not both.

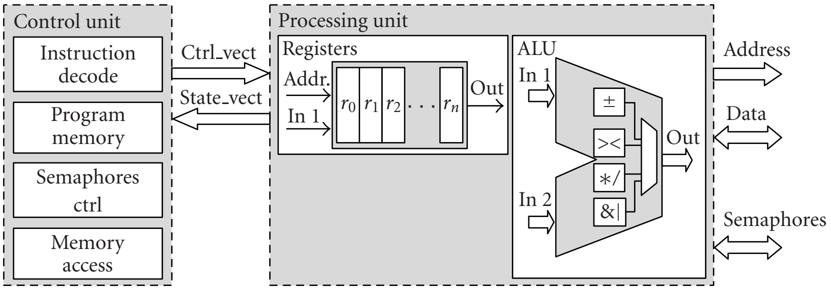


Figure 6: Internal architecture of the processing and control units.

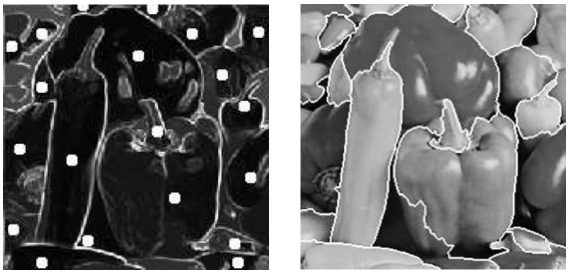


Figure 7: The "peppers" image: (a) gradient and manually placed markers; (b) continuous watershed obtained with massive marching on four processing units.

Semaphores and arbitrage

Every operation asking to access to a shared block uses the semaphores (block semaphores ctrl at Figure 6). The code that performs the semaphore-controlled access must respect the following:

```
loop :
test semaphore x         // test and lock immediately if free
if x is not free jump loop  // repeat otherwise
read/write               // access to the memory
release semaphore x      // release the semaphore
```

Whenever a semaphore is tested, it is (by the same instruction) immediately locked, provided that it was free. If not, the test is repeated as long as the semaphore can be allocated to the asking processor. After the reading/writing, the semaphore is released. The semaphores are invisible to the user provided that the compiler generates the corresponding code.

Whenever a simultaneous access to a shared block occurs, an arbitrage is used to prevent conflicts. The arbitrage is a standard block that makes part of most modern multi-processor platforms. The ideal arbitrage, usually done on the *first-come-first-served* basis, and often realized as a finite state machine, is quite costly in terms of the silicium surface. We can benefit from the randomness of the asynchronous execution, limiting the likelihood of simultaneous accesses, and saving the space by using a simple arbitrage assigning the processors an uneven priority. Obviously, this is only possible up to a certain number of processors however. In this paper, we have evaluated the feasibility by measuring the activity up to four processors (see Figure 8 showing the activity distribution).

Data memory

A low data dependency that characterizes the level set family algorithms permits to use a simple global shared memory management, being referred to in the literature as *weak consistency* model, introduced in [69]. The weak consistency is characterized by three conditions (cf. [70]). (i) Before a READ or WRITE access for any processor is allowed, all synchronizations must be achieved. (ii) Before a synchronization access is allowed, all previous READ or WRITE accesses must be achieved. (iii) Synchronization accesses are sequentially consistent with respect to each other. Note that no condition concerns the order in which the accesses are performed. See [70] for details and comparison with other consistency models.

The synchronization points are imposed by the iterative nature of the algorithms. All active points must be processed (in arbitrary order) in one iteration, before the following iteration can start. This is ensured on this architecture by the fact that the data to process are read from one memory page, and the results are written to the other. As soon as all the points in one iteration are processed (all READ and WRITE accesses are achieved), the roles of the pages PAGEs(i), i = 0,1, switch. Switching the roles of the memory pages represents the synchronization.

This architecture is conceived as scalable. According to the computational power required by a given application, one can use more or fewer processing units. It follows from Figure 3 that the highest data traffic concentrates on the shared memory blocks. Thanks to the nature of the code, the reading and writing directions on both data and active points memory blocks are separated into two one-directional channels. The results of the previous iteration (values u^{n-1}) are read from one page and the new values (u^n) are written to the other. This corresponds perfectly to the weak consistency model.

Active points memory

The READ and WRITE accesses to perform on the image data are controlled by data stored in the active points memory. It is organized in two pages. One page contains points

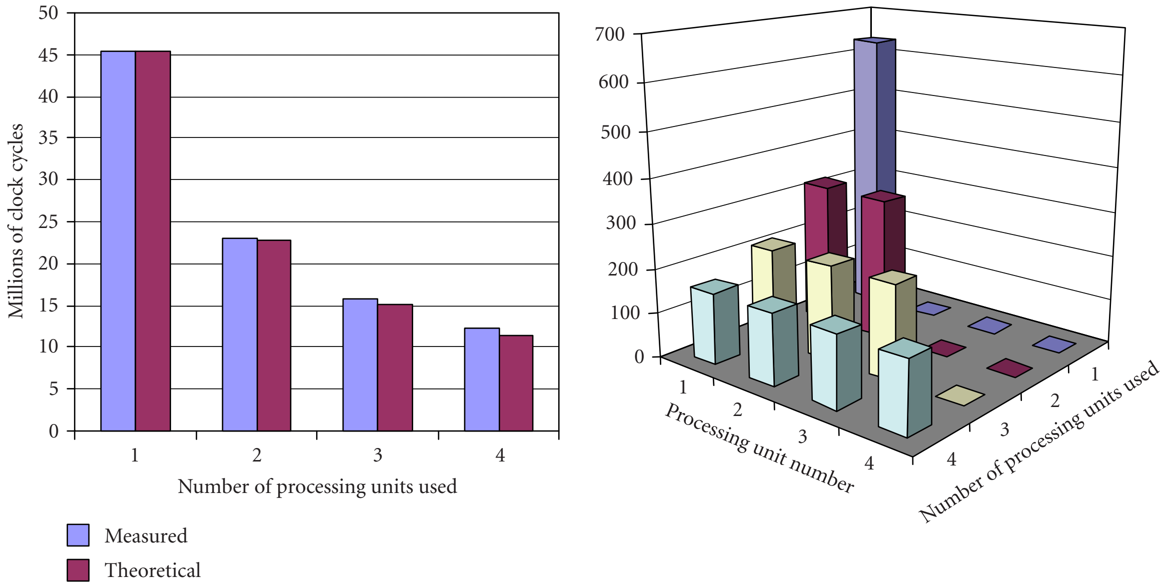


Figure 8: (a) The execution time of the algorithm in function of the number of parallely working processing units. (b) The activity load distributed over several processing units, number of points processed by every processing unit (X thousands).

to process in the current iteration. This page is progressively emptied as the data are processed. The second page contains points to process in the next iteration. It is progressively fed with data. The emptiness of one page represents the synchronization point. The roles of the pages (for both data and active points memory) switch. The emptiness of both active points pages represents the end of the algorithm.

The reading/writing direction to the data and active points memory blocks is controlled by using a boolean variable *switch* which commutes at the end of every iteration. For the sake of universality, it is left to the programmer's responsibility to control the reading.

Thanks to the fact that the processing order is indifferent, this memory can be implemented by using two LIFOs. Compared to a FIFO, using LIFO eliminates the transport delay.

For most applications, the reading should always be done on data page(*switch*) and LIFO(*switch*) and writing on page($\overline{switch}$) and LIFO($\overline{switch}$). The binary *switch* value can be derived from the zero bit of the iteration number n.

Flags

The labels and flags are similar to the data memory with a smaller word size. The labels and flags are available to the programmer for an additional algorithm control and region propagation.

4. PERFORMANCE EVALUATION

The performance of this architecture has been tested by running two different types of PDE-based algorithms: a continuous watershed and an object-tracking application.

The objective of the watershed computation is to justify the choice to use an MIMD architecture by testing whether the overall computational effort is uniformly distributed over all the processors used. The objective of the tracking application (cf. Section 4.2), is to evaluate the overall bandwidth of the architecture, and the capability to run a computationally expensive application in real time.

4.1. Evaluation test 1: A continuous watershed implementation

Recall that, in terms of PDEs, watersheds can be obtained by calculating a weighted distance function to a given set of sources, corresponding to the markers [34], while propagating simultaneously the labels

$$\|\nabla u(x, y)\| = \frac{1}{\|\nabla I\|}. \qquad (1)$$

Recall that the set of sources must be identical with the set of local minima in the image, as shown in [71]. The distance function was computed in a semiparallel way, on four parallely operating processing units, from a manually placed set of markers, see Figure 7.

Figure 8b shows the execution time (in terms of total clock cycles against the number N of processing units operating in parallel). The obtained number of clock cycles corresponds to the theoretical number of clock cycles calculated as $\mathrm{clk}_N = \mathrm{clk}_1/N$. The measured execution time (expressed in terms of clock cycles) slightly exceeds the theoretical value because of the access to the shared blocks (memory, LIFO), controlled by a semaphore. Figure **??** gives the computational load distributed over the processing units in function of the number of processing units used. The computational load is expressed in terms of number of points processed by every processing unit. If only one unit is used, the total computational load is covered by this unit. If more processing units operate in parallel, the load is uniformly distributed.

Table 1: The obtained bandwidth for watershed computation versus other platforms.

Platform	Frequency	Bandwidth (10^3points/s)
Proposed MIMD architecture Four RISC processors	120 MHz FPGA	2 610
PC with P4 Win 2000	1.6 GHz	827
IPAQ with Xscale WinCE	400 MHz	120
IPAQ with Strong ARM WinCE	200 MHz	50

Table 1 compares the bandwidth of weighted distance computation with simultaneous propagation of source labels, obtained by using massive marching implemented on various platforms. The bandwidth is computed as the number of points in the image divided by the execution time. The execution time of the proposed MIMD architecture was obtained by counting the clock cycles during simulation (HandelC code). The execution time obtained on a PC/P4, IPAQ/Xscale and StrongARM corresponds to the processor time spent in the process (programmed in C).

Note that the bandwidth of every given architecture is somewhat lesser than the theoretical bandwidth because some points are activated several times. The computation complexity of massive marching is roughly $\mathcal{O}(N)$, with N being the number of points in the image. It exceeds N by the number of reactivated points because of using a nonequidistant propagation front.

4.2. Evaluation test 2: Object-tracking application

To test the performance of this architecture, we use a model-free, gradient-based object-tracking algorithm proposed in [72].

4.2.1. A gradient-based attraction field

Consider an image $\mathcal{I}$ and some gradient of $\mathcal{I}$, $g = \nabla\mathcal{I}$. Let

$$g_K = g * K, \tag{2}$$

where K is some triangular window $\mathbb{Z}^2 \to \mathbb{R}^+$, such that

$$K(x,y) = \begin{cases} 1 - \alpha\left(x^2+y^2\right)^{1/2} & \text{if } \left(x^2+y^2\right)^{1/2} < \dfrac{1}{\alpha}, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Note that in the signal processing domain, convoluting with such a window is a frequency filter. However, filtering is not the objective here.

∇g_K represents a gradient-dependent integrator with interesting properties. Generally, the evolution of a curve $\mathcal{C}$ writes

$$\frac{\partial\mathcal{C}}{\partial t} = \mathcal{F}\vec{n}, \tag{4}$$

where$\vec{n}$is the normal vector to $\mathcal{C}$, and$\mathcal{F}$represents the motion speed. For the contour-based tracking, we propose

$$\mathcal{F} = \nabla g_K. \tag{5}$$

It can be shown (by approximating g in (2) by a Dirac impulse δ, and computing $\mathcal{F}$ in (5) in a discrete form) that ∇g_K is a bidirectional integrator pointing towards the crest of the gradient g from both sides.

The advantage of using a bidirectional integrator is twofold: (i) it allows the contour to converge towards the gradient maximum from both sides, and (ii) it eliminates the necessity to use a constant one-directional attraction force there, where the data is zero. This fact eliminates the problem of local breaches in the gradient, often introducing leakage in object reconstruction. Attempts to alleviate this problem were made in [73] introducing a viscous watershed capable to slow down the propagation in such narrow openings. Although the leakage could probably be alleviated by using curvature, the leakage problem does not occur when using ∇g_K, since on zero gradient the contour does not move.

Let ϕ represent some feature of the object to track. Supposing that this feature is unstable in time, or perturbed by external phenomena, one may need to employ an additional cue to enhance the stability. Natural gesture speed is one of the possible cues to track individuals. This fact is also used in defining the capture range of the contours. Suppose that the maximum interframe displacement of the object is bounded by D. This information should be taken into account by letting $\operatorname{supp}\{(x,y) \mid K(x,y) > 0\}$ be a circle of radius D, generating a nonzero attraction field in a narrow zone around the contour. Hence, a convenient value of α in (3) is $a = 1/D$.

Indeed, as the attraction force stops on the zero crossing of the gradient, its principle is similar to the Haralick [74] edge detector, which detects edges on zero crossing of the second derivative of $\mathcal{I}$ in the gradient direction. Kimmel and Bruckstein in [75] reformulate the Haralick edge detector in terms of the level set framework and shows how it can be combined with additive constraints to segment images. As stated before, our objective is the contour-based object tracking. Whereas various motion predictors can be used to predict the displacement direction according to the past, arbitrary deformations of the object give birth to a displacement field with locally varying direction. Any contour-based tracking must therefore be able to handle both partially forward and backward displacements of the contour. A good overview of other existing attraction vector fields can be found in [76].

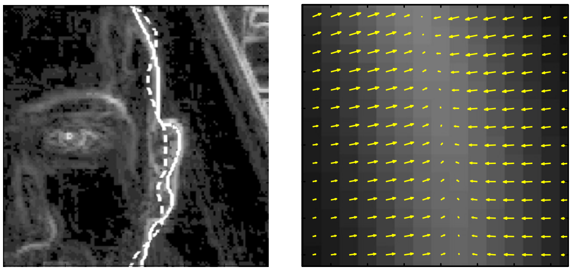


Figure 9: (a) The initial (dashed) and final (solid line) position of the contour, and (b) zoom on the attraction force field F.

4.2.2. *Application*

By integrating (4), the current contour C^n of the object is obtained by using the attraction field g_K^n generated by the current frame $\mathcal{I}^n$, and the contour C^{n-1} in the previous frame (cf. Figure 9):

$$C^n = \lim_{T\to\infty} \int_0^T \nabla g_K^n(C)\vec{n}\,dt + C^{n-1}, \tag{6}$$

$$\text{with } C(t=0) = C^{n-1}, \tag{7}$$

where $g_K^n = g * K$,

$$g(p) = \frac{\nabla_{\text{Lab}} I(p)}{1 + d_{\Omega|\phi}(I(p))}. \tag{8}$$

The ∇_{Lab} denotes the gradient on the Lab colour space. The particularity of the Lab space is that it is perceptually uniform, and ∇_{Lab} is locally Euclidean. The $d_{\Omega|\phi}$ denotes the distance to a given feature. We use a feature based on the skin chroma. We take $\Omega' \equiv \text{HLS}$, and $\phi = \{x \in \text{HLS} | x_H \in [-20^o, 50^o]\}$. This feature is only related to hue, thus the distance $d_{\text{HLS}|\phi}$ is the angular distance d^α to the skin chroma ϕ. The size of the triangular window K is ten pixels, that is, $\alpha = 0.1$, calculated from a natural gesture speed as seen by our camera.

Initialization

The description of the initialization of the tracking is outside the scope of this paper. It can be successfully done by combining several features, see for example [77], using the face colour and shape or [78] combining the colour and motion (in a car application, no perturbing motion is present in the background before the car runs).

4.2.3. *Implementation*

In the following, we outline the details concerning the implementation of the object tracking on the proposed architecture.

This architecture has been simulated using the HandelC programming language. The control units have been replaced by a pipelined model controlling each processing unit, equipped with a fully functional ALU realizing the basic arithmetic/logic operations in fixed-point precision, and equipped with a set of registers. The algorithms have been hardcoded in the control units in HandelC instructions. Note that every HandelC instruction is executed in one clock cycle.

Table 2: Frame parameters.

Frame size (X × Y)	324 × 428
Number of points in the frame	138 672
Frames per second	15
Data flow (points per second)	2 080 080

Application parameters

The video stream contains 15 frames per second, each 324 × 428 pixels, giving total data flow $2.08 \cdot 10^6$ pixels per second (cf. Table 2).

The narrowband width has been set to 20 points (ten to each side of the contour) and the mean length of the contour of the face (cf. Figure 10) to track is approximately 600 points, giving in average 12 000 active points to update per iteration, see Table 3.

The above given face tracking application requires 25 iterations in every frame for the contour to adapt itself to the new position of the face. (We consider that natural gesture speed, camera resolution, and distance to the face limit the interframe displacement of the drivers face to approximately 10 pixels.) Every five iterations, the narrowband needs to be reinitialized (cf. Table 4).

Instruction count for various algorithm steps

The construction of the attraction force field requires one convolution (cf. (2)). An $N \times N$ fast 2D convolution can be efficiently implemented by a serie of $2N$ 1D FFT applied to the columns and rows, N^2 multiplications, and a series of $2N$ 1D IFFT. Efficient algorithms exist to perform FFT/IFFT in place, see for example [79], and modern DSPs are equipped with efficient, highly optimized blocks calculating fast the FFT, for example [80].

We suppose that the convolution is computed on a companion chip. In the following, we focus on the implementation of the level-set-based part of the application, that is, the (i) initialization and construction of the narrowband, (ii) contour evolution.

The gradient can be calculated with two additions and two divisions (if central differences are used). The attraction force ∇g_K calculated on the entire frame requires 277, 344 additions and as many multiplications (cf. Table 5). The construction of the narrowband, by using massive marching, requires two steps: (i) the *interpolation* to initialize the contour can be done with 4 additions per point and (ii) the *propagation* of the distance function requires 5 additions and 6 multiplications per point. Performed twice (Jacobi and Gauss-Seidel steps) on 12 000 points (narrowband size from Table 3) gives 216 000 additions and 144 000 multiplication required to construct the narrowband. The narrowband is reconstructed five times per frame, giving the level set inherent computational effort of 1 080 000 additions and 720 000 multiplications per image frame.

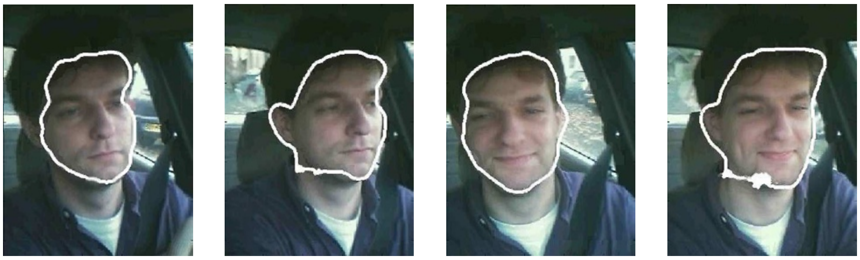


Figure 10: Contour tracking applied to driver's face extraction, using the weighted gradient (skin chroma being the feature of interest). Randomly chosen images from a video sequence.

Table 3: Narrowband parameters.

Narrow bandwidth (points)	20
Approximate mean contour length (points)	600
Number of points in the narrowband	12 000

Table 4: Object-tracking application parameters.

Number of iterations before reinitialization	5
Reinitializations per frame	5
Number of iterations per frame	25

The actual curve evolution involves several steps: (i) the evolution speed $\mathcal{F}$ requires 3 addition and 4 multiplications (including the gradient of the distance function U), (ii) the integration is done in one additions and one multiplication, giving in total 4 additions and 5 multiplications per point. Multiplied by 12 000 points in the narrowband (48 000 additions and 60 000 multiplications) and by 25 iterations per frame gives $1.2 \cdot 10^6$ additions and $1.5 \cdot 10^6$ multiplications per frame. The total application effort is $2.56 \cdot 10^6$ additions and $2.50 \cdot 10^6$ multiplications per frame, representing in total 75.8 MFLOPS to run in real time.

Table 6 presents the lower limits of the bandwidth obtained for different steps of the object-tracking application. The computation of the gradients ∇g_K and ∇u requires the same elementary operations (differences and extrema computation on the neighborhood), and presents obviously the same bandwidth $19.3 \cdot 10^6$. The limiting factor in this case is the neighborhood extraction from the input image. We have obtained the same bandwidth estimation for the integration step. The integration does not read the neighborhood (already stored in the registers) but only writes the integration result. Its performance can sometimes be limited by the bandwidth of the foregoing step.

The bandwidth $2.61 \cdot 10^6$ points/s, obtained for the narrowband construction, includes the detection of the initial contour position by interpolation and the propagation of the distance function.

We evaluate the performance of the architecture by computing the processing time of the each algorithm stage as a function of the number of processed points and these measured worst-case bandwidths. The processing time of all the steps is obtained by multiplying the worst-case bandwidth, the number of iterations, and the number of the points to process.

The sum of the processing times of individual steps gives the frame-to-frame processing time $6.18 \cdot 10^{-2}$ seconds, corresponding to 16.3 processed frames per second.

The performance, outlined in Table 7, compares the execution time of one iteration of the above-detailed object-tracking application on this architecture compared to similar results obtained on other platforms reported in the literature.

The nVIDIA GeForce2 graphic card, see [16], operates in integer accuracy, and is therefore less useful for algorithms requiring multiple iterations. The application running on PC P4, see [81], was implemented by using the additive operator splitting (AOS) scheme, permitting greater integration step, and requiring thus fewer iterations.

4.3. *Power assessment*

As the silicium surface on FPGAs continues to grow (to become comparable to ASICs), the computational power is no longer a limiting factor for the design. Instead, the preoccupations concern more and more the energy dissipation and the system autonomy.

The energy budget of some algorithm can be characterized by the energy necessary to execute the elementary operation multiplied by the number this operation is executed. Suppose that this algorithm is to be executed in a limited time. A parallel execution (provided that the algorithm is parallelizable) will allow to reduce the clock frequency (compared to the clock frequency of the sequential implementation) and reduce the energy budget of the elementary operation.

Though it is important to take into account the energy considerations as soon as possible during the design, at this development stage, it is still difficult to estimate precisely the power consumption. The execution of the algorithms was simulated by using a general-purpose RISC processor model. The power consumption was then estimated by using the consumption reported by various soft-core processors manufacturers: for Microblazer (Xilinx), see [82]; for ARM 9 family see [83]; and compared with typical-to-maximum thermal dissipation reported for Pentium 4 at 1.6 GHz (see [84]), compare Table 8.

Table 5: Instruction count for various steps.

Instruction count for various steps	*Additions*	*Multiplications*
Preprocessing		
∇g_K (operations per point)	2	2
Total per frame (additions, multiplication)	277 344	277 344
Construction of the narrowband		
Interpolation (additions, multiplications per point)	4	0
Propagation (additions, multiplications per point)	5	6
Total per initialization (additions, multiplication)	216 000	144 000
Total level-set-inherent computational effort	1 080 000	720 000
Curve evolution		
Evolution speed $\mathcal{F} = \nabla g_K \cdot \nabla U$	3	4
Integration (additions, multiplications per point) $U = U - (\mathcal{F}\,dt)$	1	1
Curve evolution per point (additions, multiplications)	4	5
Curve evolution per iteration (additions, multiplication)	48 000	60 000
Total curve evolution per frame	1 200 000	1 500 000
Total application per frame (curve evolution + level set inherent)	2 557 344	2 497 344
Overall real-time computational effort (FLOPS)	$75.8 \cdot 10^6$	

Table 6: The Execution Time of the Object Tracking Application.

Algorithm step	Estimated bandwidth (point/s)	Number of iterations	Number of points	Processing time (s)
Initialization				
Gradient ∇g_K	$19.3 \cdot 10^6$	1	138 672	$7.19 \cdot 10^{-3}$
Narrowband construction	$2.61 \cdot 10^6$	5	12 000	$2.30 \cdot 10^{-2}$
Evolution				
Gradient ∇u	$19.3 \cdot 10^6$	25	12 000	$1.56 \cdot 10^{-2}$
Integration u^{n+1}	$19.3 \cdot 10^6$	25	12 000	$1.56 \cdot 10^{-2}$
Total execution time (per frame)				$6.13 \cdot 10^{-2}$
Application frame processing rate (frame/s)				16.3

Table 7: The execution time of one iteration, compared to similar algorithms on other platforms.

Platform	Frequency	Execution time for one iteration (ms)
Proposed MIMD architecture Four RISC processors	120 MHz/FPGA	1.25
Graphic hardware nVIDIA GeForce2	250 MHz	4
PC with P4/Win 2000	1.6 GHz	19.1

Table 8: Comparison of power consumption.

Processor	Power consumption (W)
Microblazer / Xilinx	0.11
ARM9 / ARM	0.14
Pentium4 (1.6 GHz)/ Intel	60–75

5. CONCLUSIONS

In this paper, we present an embedded architecture for real-time image processing using level-set-based active contours. The contribution of this paper is twofold. In its first part, the text proposes a unifying insight into the level set framework from the system design point of view, to propose a unique iteration type with two different types of memory access: random memory access and sequential memory access. Then it analyzes the data flow to define, in the second part of the text, a scalable architecture fitting the real-time needs and taking into account the limited energy autonomy of embedded platforms and the silicium surface on commercially available FPGAs.

The performance of the proposed architecture has been studied on two benchmarks.

The first one, computation of a weighted distance transform with simultaneous propagation of region labels, is to verify the uniformity of the data flow and the distribution of the computational burden over all the processing units. This benchmark compares the real execution time against the theoretical execution time (obtained as the time needed by one processing unit divided by the number of processing units operating in parallel). The results show a linear increase of performance and a balanced activity at least up to four independently operating processing units.

The second benchmark implements an active-contour-based object-tracking algorithm. The purpose of this test is to evaluate the capability of this platform to run in real-time applications with intensive random memory accesses. Section 4.2.3 lists the details concerning the computational complexity of the application in terms of number of elementary operations. The simulation results show that the above-presented contour tracking application can be run on this architecture in real time, provided that the processors are clocked at 120 MHz, and one instruction executes in one clock cycle. Hence, the architecture specifications made in the first part of the text are confirmed.

The scalability of this architecture consists in replicating the processing units. Physically, their number is limited by the silicium available on the chip; and logically, by the data-flow balance on all the blocks of the architecture. A time-costly computation will allow a linear increase of the performance up to a higher number of processing units, before the busses and the memory blocks saturate. From Figure 3, it follows that the highest data flow concentrates on the READ data memory. Although it has not been used in this paper, two possible improvements will make the data flow on the individual memory blocks more uniform: (i) the entire four-neighborhood can be retrieved in one clock cycle by using another memory organization, as proposed by Noguet in [67], or (ii) the READ data memory flow can be divided by two by using a dual-port memory for the data memory pages. However, both options will lead to some increase of complexity of the switch.

REFERENCES

[1] S. Osher and J. A. Sethian, "Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations," *Journal of Computational Physics*, vol. 79, no. 1, pp. 12–49, 1988.

[2] S. Osher and R. P. Fedkiw, "Level set methods: an overview and some recent results," *Journal of Computational Physics*, vol. 169, no. 2, pp. 463–502, 2001.

[3] J. A. Sethian, *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision and Materials Science*, Cambridge University Press, Cambridge, UK, 1996.

[4] G. Sapiro, *Geometric Partial Differential Equations and Image Analysis*, Cambridge University Press, New York, NY, USA, 2001.

[5] D. Adalsteinsson and J. A. Sethian, "A fast level set method for propagating interfaces," *Journal of Computational Physics*, vol. 118, no. 2, pp. 269–277, 1995.

[6] R. Malladi, J. A. Sethian, and B. C. Vemuri, "A fast level set based algorithm for topology-independent shape modeling," *Journal of Mathematical Imaging and Vision*, vol. 6, no. 2-3, pp. 269–289, 1996.

[7] F. Precioso and M. Barlaud, "B-spline active contour with handling of topology changes for fast video segmentation," *EURASIP Journal on Applied Signal Processing*, vol. 2002, no. 6, pp. 555–560, 2002, Special Issue on Image Analysis for Multimedia Interactive.

[8] G. Cserey, C. Rekeczky, and P. Földesy, "PDE-based histogram modification with embedded morphological processing of the level-sets," *Journal of Circuits, Systems and Computers*, vol. 12, no. 4, pp. 519–538, 2003.

[9] J. Weickert, B. M. T. H. Romeny, and M. A. Viergever, "Efficient and reliable schemes for nonlinear diffusion filtering," *IEEE Trans. Image Processing*, vol. 7, no. 3, pp. 398–410, 1998.

[10] F. Catté, P.-L. Lions, J.-M. Morel, and T. Coll, "Image selective smoothing and edge detection by nonlinear diffusion," *SIAM Journal on Numerical Analysis*, vol. 29, no. 1, pp. 182–193, 1992.

[11] R. Goldenberg, R. Kimmel, E. Rivlin, and M. Rudzsky, "Fast geodesic active contours," *IEEE Trans. Image Processing*, vol. 10, no. 10, pp. 1467–1475, 2001.

[12] P. Smereka, "Semi-implicit level set methods for curvature and surface diffusion motion," *Journal of Scientific Computing*, vol. 19, no. 1-3, pp. 439–456, 2003.

[13] S. Holmgren and D. Wallin, *Performance of High-Accuracy PDE Solvers on a Self-Optimizing NUMA Architecture*, vol. 2150 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, 2001.

[14] J. A. Sethian, "Parallel level set methods for propagating interfaces on the connection machine," Department of Mathematics, University of California at Berkeley, Berkeley, Calif, USA, 1989.

[15] M. Rumpf and R. Strzodka, "Nonlinear diffusion in graphics hardware," in *Proc. EG/IEEE TCVG Symposium on Visualization (VisSym '01)*, pp. 75–84, Ascona, Switzerland, May 2001.

[16] M. Rumpf and R. Strzodka, "Level set segmentation in graphics hardware," in *Proc. International Conference on Image Processing (ICIP '01)*, vol. 3, pp. 1103–1106, Thessaloniki, Greece, October 2001.

[17] J. E. Cates, A. E. Lefohn, and R. T. Whitaker, "GIST: an interactive, GPU-based level set segmentation tool for 3D medical images," *Medical Image Analysis*, vol. 8, no. 3, pp. 217–231, 2004.

[18] C. Sigg, R. Peikert, and M. Gross, "Signed distance transform using graphics hardware," in *Proc. 14th IEEE Visualization Conference (VIS '03)*, pp. 83–90, Seattle, Wash, USA, October 2003.

[19] K. Hwang, P. S. Tseng, and D. Kim, "An orthogonal multiprocessor for parallel scientific computations," *IEEE Trans. Comput.*, vol. 38, no. 1, pp. 47–61, 1989.

[20] T. Gijbels, P. Six, L. Van Gool, F. Catthoor, H. De Man, and A. Oosterlinck, "A VLSI-architecture for parallel non-linear diffusion with applications in vision," in *Proc. IEEE Workshop on VLSI Signal Processing VII*, pp. 398–407, La Jolla, Calif, USA, October 1994.

[21] E. Dejnožková and P. Dokládal, "A parallel architecture for curve-evolution PDEs," *Image Analysis and Stereology*, vol. 22, pp. 121–132, 2003.

[22] R. Wittig and P. Chow, "OneChip: an FPGA processor with reconfigurable logic," in *Proc. IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '96)*, K. L. Pocek and J. Arnold, Eds., pp. 126–135, IEEE Computer Society, Napa Valley, Calif, USA, April 1996.

[23] Z. A. Ye, A. Moshovos, S. Hauck, and P. Banerjee, "CHIMAERA: a high-performance architecture with a tightly-coupled reconfigurable functional unit," in *Proc. 27th International Symposium on Computer Architecture*, pp. 225–235, British Columbia, Canada, 2000.

[24] Y. Li, T. Callahan, E. Dernell, R. Harr, U. Kurkure, and J. Stockwood, "Hardware-software co-design of embedded reconfigurable architectures," in *Proc. 37th Design Automation Conference (DAC '00)*, pp. 507–512, Los Angeles, Calif, USA, June 2000.

[25] P. Perona and J. Malik, “Scale-space and edge detection using anisotropic diffusion,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 12, no. 7, pp. 629–639, 1990.

[26] L. Alvarez, P.-L. Lions, and J.-M. Morel, “Image selective smoothing and edge detection by nonlinear diffusion. II,” *SIAM Journal on Numerical Analysis*, vol. 29, no. 3, pp. 845–866, 1992.

[27] S. Schüpp, *Prétraitement et segmentation d’images par mise en oeuvre de techniques basées sur les équations aux dérivées partielles : application en imagerie microscopique biomedicale*, Ph.D. thesis, Université de Caen Basse-Normandie, Caen Cedex, France, December 2000.

[28] B. Kimia and K. Siddiqi, “Geometric heat equation and nonlinear diffusion of shapes and images,” *Computer Vision and Image Understanding*, vol. 64, no. 3, pp. 305–322, 1996.

[29] T. Lindeberg, *Scale-Space Theory in Computer Vision*, Kluwer Academic, Dordrecht, The Netherlands, 1994.

[30] L. Alvarez, F. Guichard, P.-L. Lions, and J.-M. Morel, “Axioms and fundamental equations of image processing,” *Archive for Rational Mechanics and Analysis*, vol. 123, pp. 199–257, 1993.

[31] R. Brockett and P. Maragos, “Evolution equations for continuous-scale morphology,” in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing (ICASSP ’92)*, vol. 3, pp. 125–128, San Francisco, Calif, USA, March 1992.

[32] R. van den Boomgaard, *Mathematical morphology: extensions towards computer vision*, Ph.D. thesis, University of Amsterdam, Amsterdam, The Netherlands, March 1992.

[33] F. Meyer and P. Maragos, “Nonlinear scale-space representation with morphological levelings,” *Journal of Visual Communication and Image Representation*, vol. 11, no. 2, pp. 245–265, 2000.

[34] L. Najman and M. Schmitt, “Watershed of a continuous function,” *Signal Processing*, vol. 38, no. 1, pp. 99–112, 1994.

[35] A. Montanvert and J. M. Chassery, *Géométrie discrète en analyse d’images*, Hermès, Paris, France, 1991.

[36] R. Kimmel, *Curve evolution on surfaces*, Ph.D. thesis, Technion - Israel Institute of Technology, Haifa, Israel, May 1995.

[37] J. A. Sethian, “A marching level set method for monotonically advancing fronts,” *Proceedings of the National Academy of Sciences*, vol. 93, no. 4, pp. 1591–1595, 1996.

[38] M. Kass, A. Witkin, and D. Terzopoulos, “Snakes: active contour models,” *International Journal of Computer Vision*, vol. 1, no. 4, pp. 321–331, 1987.

[39] D. Terzopoulos, A. Witkin, and M. Kass, “Constraints on deformable models: recovering 3D shape and nonrigid motions,” *Artificial Intelligence*, vol. 36, no. 1, pp. 91–123, 1988.

[40] L. Cohen, “On active contour models and balloons,” *Computer Vision, Graphics, and Image Processing*, vol. 53, no. 2, pp. 211–218, 1991.

[41] V. Caselles, F. Catté, T. Coll, and F. Dibos, “A geometric model for active contours in image processing,” *Numerische Mathematik*, vol. 66, no. 1, pp. 1–31, 1993.

[42] R. Malladi, J. A. Sethian, and B. C. Vemuri, “Topology independent shape modeling scheme,” in *Geometric Methods in Computer Vision II*, vol. 2031 of *Proceedings of SPIE*, pp. 246–258, San Diego, Calif, USA, July 1993.

[43] R. Malladi, J. A. Sethian, and B. C. Vemuri, “Evolutionary fronts for topology-independent shape modeling and recovery,” in *Proc. 3rd European Conference on Computer Vision (ECCV ’94)*, vol. 800 of *Lecture Notes in Computer Science*, pp. 3–13, Stockholm, Sweden, May 1994.

[44] R. Malladi, J. A. Sethian, and B. C. Vemuri, “Shape modeling with front propagation: a level set approach,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 17, no. 2, pp. 158–175, 1995.

[45] R. Malladi, R. Kimmel, D. Adalsteinsson, G. Sapiro, V. Caselles, and J. A. Sethian, “A geometric approach to segmentation and analysis of 3d medical images,” in *Proc. Mathematical Methods in Biomedical Image Analysis Workshop (MMBIA ’96)*, San Francisco, Calif, USA, June 1996.

[46] V. Caselles, R. Kimmel, and G. Sapiro, “Geodesic active contours,” in *Proc. 5th IEEE International Conference on Computer Vision (ICCV ’95)*, pp. 694–699, Cambridge, Mass, USA, June 1995.

[47] S. Kichenassamy, A. Kumar, P. Olver, A. Tannenbaum, and A. Yezzi, “Gradient flows and geometric active contours models,” in *Proc. 5th International Conference on Computer Vision (ICCV ’95)*, pp. 810–815, Cambridge, Mass, USA, June 1995.

[48] R. Malladi and J. A. Sethian, “Image processing: flows under min/max curvature and mean curvature,” *Graphical Models and Image Processing*, vol. 58, no. 2, pp. 127–141, 1996.

[49] S. Jehan-Besson, M. Gastaud, M. Barlaud, and G. Aubert, “Region-based active contours using geometrical and statistical features for image segmentation,” in *Proc. IEEE International Conference in Image Processing (ICIP ’03)*, vol. 2, pp. 643–646, Barcelona, Spain, September 2003.

[50] L. Alvarez, J. Weickert, and J. Sánchez, “A scale-space approach to nonlocal optical flow calculations,” in *Proc. 2nd International Conference on Scale-Space Theories in Computer Vision (Scale-Space ’99)*, pp. 235–246, Corfu, Greece, September 1999.

[51] J. B. T. M. Roerdink and A. Meijster, “The watershed transform: definitions, algorithms and parallelization strategies,” *Fundamenta Informaticae*, vol. 41, no. 1-2, pp. 187–228, 2000.

[52] E. Dejnožková, *Architecture dédiée au traitement d’image basé sur les équations aux dérives partielles*, Ph.D. thesis, Ecole Nationale Supérieure des Mines de Paris, Paris, France, March 2004.

[53] J. A. Sethian, “Fast marching methods,” *SIAM Review*, vol. 41, no. 2, pp. 199–235, 1999.

[54] J. A. Sethian, “Level set methods and fast marching methods,” Tech. Rep., Department of Mathematics, University of California, Berkeley, Calif, USA, 1999. http://math.berkeley.edu/∼sethian/level_set.html.

[55] S. Kim, “An $\mathcal{O}$(N) level set method for eikonal equations,” *SIAM Journal on Scientific Computing*, vol. 22, no. 6, pp. 2178–2193, 2001.

[56] M. J. Flynn, “Very high-speed computing systems,” *Proc. IEEE*, vol. 54, no. 12, pp. 1901–1909, 1966.

[57] R. Cypher and J. L. C. Sanz, “SIMD architecture and algorithms for image processing and computer vision,” *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, no. 12, pp. 2158–2174, 1989.

[58] A. Gibbons and W. Rytter, *Efficient Parallel Algorithms*, Cambridge University Press, Cambridge, UK, 1988.

[59] A. Broggi, G. Conte, F. Gregoretti, C. Sansoè, and L. M. Reyneri, “The Paprica massively parallel processor,” in *Proc. 1st IEEE International Conference on Massively Parallel Computing Systems (MPCS ’94)*, pp. 16–30, Ischia, Italy, May 1994.

[60] A. Manzanera, “Morphological segmentation on the programmable retina: towards mixed synchronous/asynchronous algorithms,” in *Proc. 6th International Symposium on Mathematical Morphology (ISMM ’02)*, H. Talbot and R. Beare, Eds., pp. 389–399, Sydney, Australia, April 2002.

[61] T. Le, W. Snelgrove, and S. Panchanathan, “SIMD processor arrays for image and video processing: a review,” in *Multimedia Hardware Architectures*, vol. 3311 of *Proceedings of SPIE*, pp. 30–41, San Jose, Calif, USA, January 1998.

[62] M. Maruyama, H. Nakahira, T. Araki, et al., “A 200 MIPS image signal multiprocessor on a single chip,” in *Proc. 37th IEEE International Solid-State Circuits Conference (ISSCC ’90)*, pp. 122–123, San Francisco, Calif, USA, February 1990.

[63] T. Minami, R. Kasai, H. Yamauchi, Y. Tashiro, J. Takahashi, and S. Date, "A 300-MOPS video signal processor with a parallel architecture," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 12, pp. 1868–1875, 1991.

[64] R. Duncan, "A survey of parallel computer architectures," *IEEE Computer*, vol. 23, no. 2, pp. 5–16, 1990.

[65] E. De Greef, F. Catthoor, and H. De Man, "Mapping real-time motion estimation type algorithms to memory efficient, programmable multi-processor architectures," *Microprocessing and Microprogramming*, vol. 41, no. 5-6, pp. 409–423, 1995.

[66] A. Moga, T. Viero, B. Dobrin, and M. Gabbouj, "Implementation of a distributed watershed algorithm," in *Mathematical Morphology and Its Applications to Image and Signal Processing*, pp. 281–288, Kluwer Academic, Dordrecht, The Netherlands, 1994.

[67] D. Noguet, *Architectures parallèles pour la morphologie mathématique géodésique*, Ph.D. thesis, Institut National Polytechnique De Grenoble, Techniques de l'Informatique et de la Microélectronique pour l'Architecture des ordinateurs, Grenoble, France, Janvier 2002.

[68] A. Bieniek, *Divide-and-Conquer Parallelisation Methods for Digital Image Processing Algorithms*, vol. 10 of *VDI Fortschritt-Berichte*, VDI Verlag, Düsseldorf, Germany, 2000.

[69] M. Dubois, C. Scheurich, and F. Briggs, "Memory access buffering in multiprocessors," in *Proc. 13th Annual International Symposium on Computer Architecture (ISCA '86)*, pp. 434–442, Tokyo, Japan, June 1986.

[70] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy, "Memory consistency and event ordering in scalable shared-memory multiprocessors," in *Proc. 17th Annual International Symposium on Computer Architecture (ISCA '90)*, pp. 15–26, Seattle, Wash, USA, May 1990.

[71] F. Meyer and P. Maragos, "Multiscale morphological segmentations based on watershed, flooding, and eikonal PDE," in *Proc. 2nd International Conference on Scale-Space Theories in Computer Vision (Scale-Space '99)*, M. Nielsen, P. Johansen, O. F. Olsen, and J. Weickert, Eds., vol. 1682 of *Lecture Notes in Computer Science*, pp. 351–362, Springer, Corfu, Greece, September 1999.

[72] P. Dokládal, R. Enficiaud, and E. Dejnožková, "Contour-based object tracking with gradient-based contour attraction field," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing (ICASSP '04)*, vol. 3, pp. 17–20, Montreal, Quebec, Canada, May 2004.

[73] F. Meyer and C. Vachier, "Image segmentation based on viscous flooding simulation," in *Proc. 6th International Symposium on Mathematical Morphology (ISMM '02)*, vol. 2, pp. 69–77, Sydney, Australia, April 2002.

[74] R. Haralick, "Digital step edges from zero crossing of second directional derivatives," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 6, no. 1, pp. 58–68, 1984.

[75] R. Kimmel and A. Bruckstein, "On edge detection integration and geometric active contours," in *Proc. 6th International Symposium on Mathematical Morphology (ISMM '02)*, vol. 2, Sydney, Australia, April 2002.

[76] C. Xu and J. L. Prince, "Snakes, shapes, and gradient vector flow," *IEEE Trans. Image Processing*, vol. 7, no. 3, pp. 359–369, 1998.

[77] K. Sobottka and I. Pitas, "Extraction of facial regions and features using color and shape information," in *Proc. 13th IEEE International Conference on Pattern Recognition (ICPR '96)*, vol. 3, pp. 421–425, Vienna, Austria, August 1996.

[78] A. Nayak and S. Chaudhuri, "Self-induced color correction for skin tracking under varying illumination," in *Proc. IEEE International Confererence on Image Processing (ICIP '03)*, vol. 3, pp. 1009–1012, Barcelona, Spain, September 2003.

[79] V. Veselý, "Fast Algorithms of Fourier and Hartley Transform and their Implementation in MATLAB," http://citeseer.ist.psu.edu/vesely98fast.html.

[80] H. Karner, M. Auer, and C. Ueberhuber, "Optimum complexity FFT algorithms for RISC processors," Tech. Rep. AURORA TR1998-03, Institute for Applied and Numerical Mathematics, Technical University of Vienna, Vienna, Austria, 1998.

[81] S. Osher and N. Paragios, Eds., *Geometric Level Set Methods in Imaging, Vision and Graphics*, chapter 2, Springer, New York, NY, USA, 2003.

[82] http://www.eece.unm.edu/xup/microblazeppc.htm.

[83] http://www.arm.com/miscPDFs/4491.pdf.

[84] http://support.intel.com/design/pentium4/datashts/24919805.pdf.

Eva Dejnožková is a research engineer with the Commissariat à l'Energie Atomique à Saclay, France, which she joined in 2004. She graduated with the highest degrees from the West Bohemia University, Pilsen, Czech Republic, in 1999, as an engineer specialized in industrial electronics. Afterwards, she specialized in hardware architecture for image processing, and obtained her Ph.D. degree from the School of Mines of Paris, France. She obtained a special distinction of the rector of the West Bohemia University. Her research interests include image processing and compression, and embedded and mobile computers architecture for image processing and compression.

Petr Dokládal is a research engineer at the Centre of Mathematical Morphology, the School of Mines in Paris, France. He graduated from the Technical University in Brno, Czech Republic, in 1994, as a telecommunication engineer and received his Ph.D. degree from the University of Marne la Vallée, France, in general computer sciences, specialized in image processing. His research interests include medical imaging, image segmentation, object tracking, and pattern recognition.