Parallel Asynchronous Computations for Image Analysis

Rapport Interne n° 2001/22
Technical Report n° 2001/22

Laboratoire Heudiasyc UMR CNRS 6599

Bertrand Ducourthial
Heudyasic UMR CNRS 6599
Université de Technologie de Compigne,
BP 20529, 60205 Compigne, France,
Bertrand.Ducourthial@utc.fr

Alain Mérigot
IEF UMR CNRS 8622
Université de Paris Sud,
bt. 220, 91405 Orsay, France
am@ief.u-psud.fr

May 2001

Abstract

Associative nets is a well suited parallel computing model for image analysis. It is based on the simple data-parallelism paradigm, and also provides special features, such as graph based data structures to handle irregular data, virtual data-structures to ease hierarchical image descriptions, and specific primitives (`dirassoc`) to compute on the inter-pixels relation graph. It has been implemented either in hardware or in software, and has been used to simply program many fundamental image analysis algorithms.

For implementation purpose, the `dirassoc` computing primitive performs asynchronous local computations until stability. Asynchronism has many advantages for hardware (speed, power consumptions, chip size `etc.`) as well as in software (less synchronization barriers). However, the `dirassoc` computing primitive must be used either with particular operator or on particular graphs, to obtain the convergence on a deterministic result of the asynchronous computations. In [6], we enlarged the class of authorized operators, defining the r-operators. These operators extend the possibility of the asynchronous computations.

In this paper, we emphasize on the interest of the r-operators and of the asynchronous computations for image analysis algorithms. We give applications in distance transforms, contour closing, Voronoï segmentation, watershed segmentation and mathematical morphology. All these algorithms take benefit of the asynchronous computations, not only for their implementations, but also directly at the programming level. Hence, we show that asynchronous computing primitives are powerful tools for image analysis.

keywords: Image analysis, Parallel computations, Asynchronous computations, Distance transformation, Contour closing, Voronoï segmentation, Watershed segmentation, Mathematical morphology.

1 Introduction

1.1 Image analysis

Several properties characterize image processing. First, an image represents a large amount of data, and depending on the applications, it must be processed under strict speed constraints. Second, the elementary data is very small (a pixel); in most cases, it only requires a single byte to be properly coded. Third, at least in the early processing stages, a rigid local geographical relationship exists between these data, which are arranged according to a 2D mesh.

According to these characteristics, classical parallel architectures, that are mainly tuned towards the flexibility and the computing power, required by numerical analysis techniques, are not perfectly suited for image processing analysis, and a scheme based on a large number of small processing elements could probably be more efficient.

Indeed, according to these considerations, there have been several parallel computer architectures proposed for image processing. Thanks to the geographic local organization of the pixels in an image as a two dimensional mesh, and to the regularity of most low-level computations, mesh-based parallel architectures are quite popular for image analysis applications [12, 16]. Indeed they allow efficient implementations of basic neighbor-based primitives (convolution, filtering, *etc.*).

However, the situation is more complex when dealing with higher level image analysis algorithms. These algorithms are characterized by several peculiarities.

- First, as their objective is mainly to find structural information in an image to ease higher level recognition phases, they exhibit much more irregular and data dependent behavior than low-level operations. Most information is rather given by the complete inter-pixel relationship, and not only by one pixel neighborhood. For instance, interesting information elements may be segmented objects, contour segments, *etc.* While this kind of information needs more time to be used, it is required for many applications to exploit relational informations beyond the neighborhood of the nodes. The main problem lies in the irregular and data dependent nature of this kind of information, which cannot be efficiently exploited either by compilers or by most parallel architectures.

- Second, efficient programmability is a key issue for these applications. While low-level operations can generally be specified by a simple parameter setting, useful image analysis may require completely different methods. For instance, a vision system for smart vehicule guidance may have to use many different methods to extract useful informations, depending on the environment, the traffic, the external weather, *etc.* The situation is similar when performing video indexing or smart compression.

- Last, image analysis is presently the most complex and time consuming part in a vision chain. An obvious solution to this problem is the use of parallel processing. Technology evolution will allow an increasing complexity to be integrated within a simple chip. It is not presently clear which will be the dominating concept in terms of parallelism: SIMD (like multimedia extensions), shared or distributed memory MIMD or VLIW; most probably a combination of these schemes.

1.2 Data parallel programming and image analysis

According to these considerations, we point out the need of a *high-level* programming model that should be able to cope with the the irregular and data dependent operations of image analysis, while facilitating algorithmic description and allowing an efficient implementation on a large set of parallel architectures.

Defining a programming model means to specify what is the programmer view of the information (*data structure*), and how it will manipulate it (*computing primitives*). Both of them should be concurrently *simple* and *powerful*. Simplicity means that they should rely on a limited set of clear concepts. While this may facilitate an efficient use of the model by the programmer and reduce the learning curve, this is an absolute requirement for any implementation, either in software (programming libraries or even compilers) or in hardware.

Powerfulness implies that these concepts must allow an efficient description of most of the *objects* used in image analysis (images, regions, contours, hierarchies, *etc.*). In terms of algorithms, it means that any algorithm may be described by chaining the programming primitives of the model.

In some sens, we want to define an equivalent for image analysis to the widely used *vector model* used in scientific computing. This model relies on a limited set of simple data structures (scalars, vectors, dense or sparse matrices) and some programming primitives that operate in parallel on the data object (vector and matrix operations, scatter, gather, *etc.*). Despite its simplicity, this model is able to describe any numerical algorithm. Thanks to these peculiarities, it is the fundation of several widely used software tools : numerical libraries (NAG, IMSL), programming languages (fortran 90), software prototyping tools (Matlab, SciLab), *etc.* In terms of architectures, this model is efficiently used either for *special purpose* architectures (so-called vector computers) or for general purpose parallel computer (shared memory, distributed memory, *etc.*), by means of libraries or compiling tools.

Before describing the model, we would like to emphasize on the importance of parallel aspects. Indeed extracting parallelism from a sequential description is a complex task. Even for the mostly regular problem of scientific computations, decades of research work have been necessary to obtain the not totally satisfactory present parallelizing tools. The situation is more critic when the target application is irregular (which is the case of most image analysis algorithms), and particularly when this irregularity is unpredictable, such as, for instance, image segmentation.

A good solution is to use a parallel description at the programming level. This description must be at a very high level and should not rely on any assumption on the underlying architecture, in order to cope with potentially different parallel concepts. A good candidate is data parallelism that express tranforms globally on a set of data. Indeed, provided the programmming concepts are adapted to the application, such a description can lead to compact algorithmic descriptions. More, generating optimized either sequential or parallel code from this kind of descriptions is much simpler that parallelizing a sequential data manipulation. And worth of nothing is the fact that data-parallel programming is easy to understand and use, which allows a large acceptation by image analysis designers (see examples in § 3).

1.3 Using graphs to describe image features

Existing data-parallel environments are not sufficient to deal with irregular objects in the images. So we need to introduce new tools. To cope uniformely with the varied features that are used in image analysis, graphs based representations are good candidates. Indeed, most image analysis algorithms aim to find pixels aggregates that describe meaningful *features* in an image. These structural elements can be expressed by using a graph that represent *relations* between pixels. For instance, we can use a graph to describe regions in an image by linking together neighboring pixels belonging to the same region. While providing a flexible way to describe any kind of interpixel relations, this description can be formally uniform for a large range of features.

We have proposed the associative net model [14], a graph based computing model well designed for image analysis [7]. Its main purpose is to provide simultaneously an expressive high level programming model and a potentially efficient hardware (and software) implementation. It relies on a basic *mesh* graph, a 8-connected 2-dimensional grid that represents the neighborhood of the pixels.

This model can efficiently exploit structured information within an image, by representing it by means of a subgraph of the mesh that interconnects subsets of adjacent pixels (see § 2.1). It can also be hierarchically defined and this way describe higher level features in an image [15] (see § 2.2).

1.4 Using graphs to specify computations

Besides its usefulness as a structural description tool, the associative net model proposes to use graphs as a support for computational primitives. Some specific computing primitives, such as `dirassoc`, will gather information along the edges of the graph (see § 2.3).

The associative nets model is a superset of the mesh based scheme as `dirassoc`s allow to efficiently implement any kind of neighbor based computations. If the subgraph is regular, it is possible to perform for instance an image shift, but the local manipulation of the communication graph gives an extra flexibility.

Hence, the specificity of the model relies on global operations over graphs. Global computations can be performed over any subgraph of the mesh, such as: the whole mesh itself, graphs with regular connected sets (*e.g.* chess-boards), graphs with massive connected sets (that may represent regions) or in the shape of wires (for contours). For instance, if the considered graph has connected sets that represent regions (*i.e.* pertinent connected sets) of an image, it is possible to compute a global measure of the regions, whatever their shapes, and to get informations like area, average grey level, bounding box and so on by considering appropriate operators and data.

1.5 Using asynchronous computations over a graph

The `dirassoc` computing primitive gather some data along the edges of some graphs (subgraph of the mesh). Such a global computation is performed by some local-to-the-node computations. A local computation is performed on each node v by `assoc`iating the data of the `dir`ect ancestors of v using a given operator. Each node then builds a new data, which will be used by the direct descendant nodes for their own local computations.

As explained in § 2.3, the `dirassoc` primitive can be implemented by means of *asynchronous* local computations, until the whole network is stabilized (*i.e.* all nodes values are stable). Well known conditions for termination and deterministic result of asynchronous computations have been extended in [6]. Hence, a large spectre of operators (called r-operators, see § 2.4) can be applied asynchronously in a graph.

While such a property has been first introduced for implementation purposes, this programming functionnality has appeared as an interesting tool for algorithms, especially for computations on graphs. This paper focus on the applications of r-operators and asynchronous parallel computations for image analysis algorithms. Before investigating the interest of r-operators for the algorithmic, we first point out the advantages of asynchronous computations from an implementation point of view.

- For an asynchronous computation, the data-dependent constraints are relaxed, and the control of the computation is low. Only the termination detection must be managed, instead of using synchronization barriers between the synchronous computations (implemented by means of hardware or software techniques). Depending on the implementation, a significant improvement can be obtained, due to the diminution of control (see *e.g.* [24, 19]).

- In a grid of elementary processors, an asynchronous communication requires the average communication delay, while a synchronous communication is based on a period, that is computed on the base of the worst communication delay. Hence significant improvements can be obtained either for a pure communication or for a mixed computation-communication like the `dirassoc`.

- Some electronic implementations can take benefit of the asynchronism [10, 17]. Asynchronism allows lower power consumptions, as clock picks are avoided.

1.6 Related works

Several image processing environment exists, but most of then are targeted towards low-level primitives. A similar extension to the Pacco programming environment has been proposed[3].

The associative nets model is quite simple in its definition. It allows expressiveness for algorithmic and powerful implementations, either in hardware or in software. Hardware implementations of the associative nets model has been studied toward an SIMD multicomputer based on asynchronous communications, called *associative mesh* [10, 11, 14]. A small prototype chip has been built for performance evaluation. Moreover, a parallel computing library called *ANET* — based on C++ and threads — has been designed for shared memory multiprocessors [9]. Algorithms of Section 3 can be programmed and tested with ANET. A version for clusters of workstations is currently studied, and a matlab frontal is being developped.

Several works concern the asynchronous computations for matrix-like calculations, and a lot of experimentations have shown the interest of decreasing the control for convergence speed, communication delay and synchronization overhead (see for instance [24] and the references therein). In the image analysis field, one can find similar studies in [17, 18, 19], especially for the purpose of VLSI implementation. The authors show that some low level operators can be implemented either asynchronously or synchronously. All these studies lead to the conclusion that algorithms, and especially iterative loops, needed to successively improve an intermediate solution (see § 3.3 and 3.4), are faster when based on asynchronism, either provided by an electronic support or by a parallel software environment.

To our knowledge, no general studies has been done on asynchronous image analysis algorithms. Associative nets model and r-operators allow to generalize previous results in this field, and to define new and powerful algorithms for image analysis, with an intuitive parallelization. Section 2 summaries the model and the operators. Section 3 details some algorithms.

2 Computing Model

The associative nets model is based on the data-parallelism paradigm. However, specific data structures and primitives have been added to ease the development of image analysis algorithms.

2.1 Data structures

The associative nets model relies on a network of computing nodes, the *basic graph* `anet`. While any topology can be used, such as hexagonal grid (especially in software implementations), we assume in the rest of this paper an $N \times N$ directed grid with diagonal connections, called *mesh* (see figure 1). Such a based graph plays the role of instanciation machine: any parallel variable `pvar<T> I` is composed of one local variable `I[v]` of type `T` per node v of the based network `anet`. For instance, a 256 grey level image `I` is defined by `pvar<char> I`.

The irregular structures, such as contours or regions of the images, are connected sets of nodes. Such connected sets, called `snet`, are subnets of the based network: only the links that connect the nodes of these connected sets are conserved. To code a `snet`, it is sufficient to note, on each node, the incoming links. This can be done with a word (called `mask`) of as many bits as indegree of the based graph (a byte for the mesh, see figure 1-a). Hence, any irregular structures (figure 1-b, -c and -d) as well as regular patterns (figure 1-a) are simply and efficiently coded and handled through a `pvar<mask>`.

2.2 Virtual structures

In some cases, it is interesting to work with connected sets instead of with all the nodes they belong to. For example, when regions are identified in an image, algorithms are simpler when they handle regions themselves instead of all the nodes of all the regions. Moreover, avoiding a large number of elementary computations on nodes can lead to interesting improvements. For this prupose, associative nets provide *virtual structures*. Note that virtual structures allow hierarchical descriptions in the images, which are powerful tools for image analysis [15].

Starting from an `snet` (that codes, for instance, the regions of an image), a `vnet` is built by concentrating all the nodes of each connected set (*i.e.* each region) into a single *virtual node* (one per region, see Figure 2). Moreover, as for `anet`, the `vnet`s allow to define `vpvars` (one value per virtual node, *i.e.* per connected set such as regions). Such a `vpvar` can be constructed by reduction of a `pvar`: one value per connected set is computed from the values of the nodes of each connected set (*e.g.* the maximum, see Figure 2-a). Conversely, the `diffuse` primitive allows to set a `pvar` from a `vpvar` (Figure 2-d). Finally, as in `anet` with the `snets`,

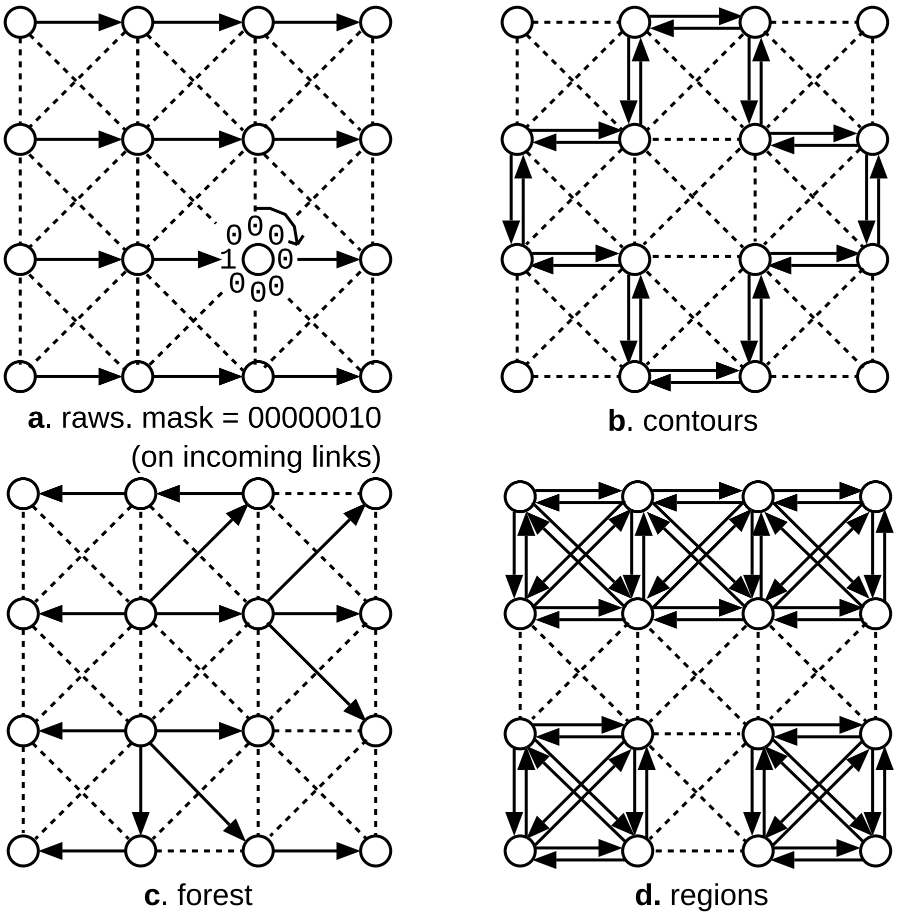

Figure 1: Regular patterns and irregular structures simply coded with a `pvar<mask>` on the mesh.

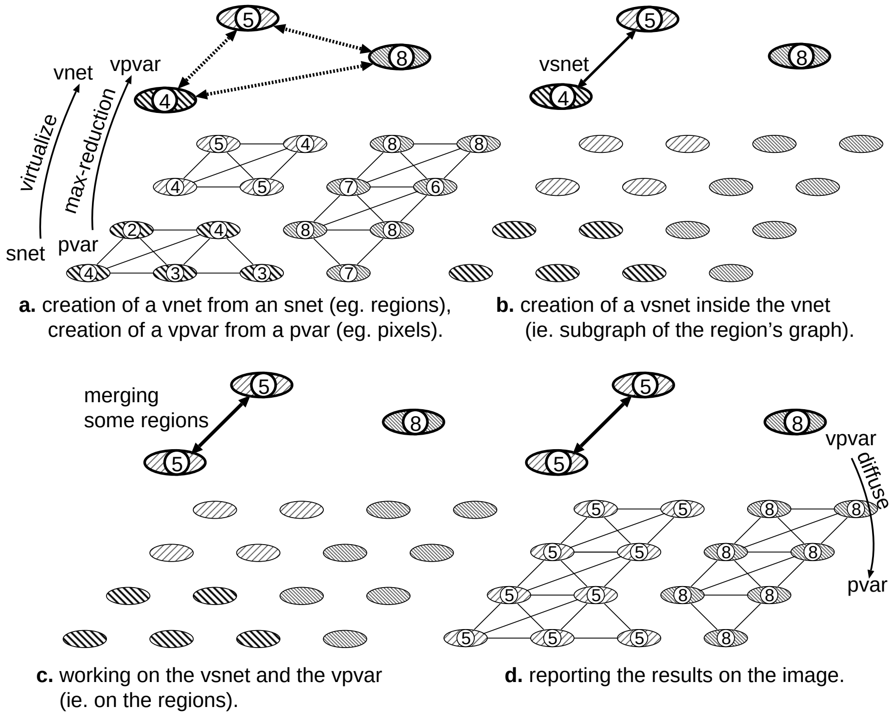

Figure 2: Working on hierarchical structures using the `vnet`s and `vsnet`s.

some subnets (called `vsnets`) of the `vnet` can be defined to compute, for example, regions that merge (Figure 2-b and -c).

2.3 Computing primitives

Associative nets provides simple computations on scalars (*e.g.* pixels) such as `I[5] = min(64,I[5])` and on `pvars` (*e.g.* the whole image) such as `I = min(64, I)`.

Specific `pvars` that code a graph (either `snet` or `vnet`) can be manipulated similarly. Specific operators exist exist that allow to manipulate the graph: return its edges (`invert`), span a forest over a the graph (`spanning-tree`), create a graph connecting to the neighbors that shares some property (`link-with-equlas`), etc[14].

Computations that need values of neighbor's nodes can be written using a pattern coded by an `snet` and the `step-dirassoc` primitive. This primitive applies a given operator on a node and some of its neighbors defined by a `snet`. For instance, the maximum of the four North, East, South and West neighbors is done with

```
pvar<mask> m = 0xaa; // grid (= 10101010)
J = max-step-dirassoc(I, m)
```

while the average of the neighborhood of each pixel (excluding itself) is computed with

```
pvar<mask> m = 0xff; // the whole mesh
J = +-step-ex-dirassoc(I, m) / 8
```

(the prefix `ex-` means that the value of the node itself is not taken into account in the computation).

However, an `snet` can code other patterns than neighborhoods, such as rows as seen in figure 1-a. To compute on such larger subnets, we introduce the `dirassoc` primitive, that cascades the `step-dirassoc`s from sources (nodes without ancestors) down to sinks (nodes without successors) in the subnet. Hence, the sum of the pixels of each row of an image `I` is performed by

```
pvar<mask> m =0x02; // rows
J = +-dirassoc(I, m)
```

This computation can also be done on forests instead of rows, as with the `snet` of figure 1-c. This appear to be a generalization of the `scans` [4].

Note that such a cascade of computations does not necessitate any synchronization between nodes of the `snets`: the results are progressing from sources to sinks, even if nodes are not synchronized. For instance, for the sum on the rows, stability can only appear after the computation of the sum has been progressed from the left to the right of the mesh, and such a progression does not require any synchronization. Hence the `dirassoc` primitive is defined as executions of local to the nodes `step-dirassoc`s until stabilization of the whole network (*i.e.* all values of the nodes are stable). As explained in § 1.5, asynchronous computing primitives have many advantages for implementations.

2.4 Operators to instanciate `dirassoc` primitive

Since there is no easy-to-use orientation in the graph, it is not realistic to organize the cascade of local computations, either in a synchronous or asynchronous way. Hence, to insure deterministic result, the operators should conform to some algebraic properties, such as associativity and commutativity. In particular, computing the minimum or the sum of some values on rows of forests (Figure 1-a end -c) leads to an unambigous and deterministic result on each node.

To insure the termination with `snet` that have cycles, operators should be idempotent. In particular, computing the minimum of some non null values in a cycle terminates, while computing the sum does not. Associative, commutative and idempotent operators (such as `min`, `max`, `or`, `and` *etc.*) are called *s-operators*. One can show that asynchronous computations stabilizes on deterministic results with such operators (see *e.g.* [8, 6]). Many image analysis applications can be written using `dirassocs` and s-operators [14, 11, 7].

In [6, 8], we introduced weaker hypothesis than associativity and commutativity for computations on general graphs. Moreover, we defined a larger class of operators for asynchronous computation, defining the *r-operators* that encompass the s-operators. For instance, the `minc` operator defined on $\mathbb{N}$ by $x, y \mapsto \mathtt{min}(x, y+1)$ is idempotent but not associative nor commutative, while still leading to the stabilization of the global computation.

Starting from s-operators such as `min`, we build an asymmetric operator by introducing an *r-mapping*, such as $x \mapsto x+1$ as with the `minc` operator. Such an operator is called an *r-operator*. More precisely, an r-operator $\triangleleft$ is built on a set $\mathbb{S}$ (*e.g.* $\mathbb{N}$) from an s-operator (*e.g.* `min`) and an homomorphism r (*e.g.* $x \mapsto x+1$), such that the following properties are fulfilled:

$$\begin{array}{rll} \text{(i)} & (x \triangleleft y) \triangleleft r(z) = x \triangleleft (y \triangleleft z) & r\text{-associativity} \\ \text{(ii)} & r(x) \triangleleft y = r(y) \triangleleft x & r\text{-commutativity} \\ \text{(iii)} & r(x) \triangleleft x = r(x) & r\text{-idempotency} \end{array}$$

Note that an s-operator is simply an s-operator where the homomorphism r is the identity mapping $x \mapsto x$. We proved that any asynchronous computation based on an idempotent r-operator terminates on a deterministic result on any graph.

The s-operators such as `min` are "symmetric"; they exploits connectivity of the subnet since a single result is obtained per connected set. With r-operators, paths of the `snet` are exploited because an r-mapping is applied each time a value passes toward a link. Hence these "asymmetric" operators allow to use the inter-pixel relationship information, coded by subgraphs of the mesh. In particular, the `minc` operator allows to compute some distances in the graph in a single asynchronous computation (see § 3.1).

Finally, two methods allow to apply non idempotent operators on cyclic graphs: directly with the specific primitive `assoc`, or with a spanning subgraph. For instance, to compute the number of pixels on the regions coded by the `snet region`, we could firt use the `spanning-tree` primitive to embed a tree in each region, then compute the sums on these trees, and finally broadcast the results on the whole regions with a maximum computation:

```
pvar<mask> forest = spanning-tree(regions);
pvar<int> nb = +-dirassoc(image, forest);
nb = max-dirassoc(nb, regions)
```

Applications of asynchronous computations based on r-operators are given in the next section.

3 Algorithms

3.1 Distance computation

Given a set of k points p_i, i in $\{0, \ldots, k\}$, in an image, the *distance transform* (DT) consists to label every pixel with its geometrical distance to the closest point p_i. This transform is the basis of several useful methods of image analysis: median axis transform, skeletization, Voronoï tesselation *etc.*

The computational cost of the Euclidean distance is often useless in most applications and simpler metrics are prefered. Popular ones are the *city bloc* metric d_4 defined by

$$d_4(p,q) = |x_p - x_q| + |y_p - y_q|$$

and the *chess board* metric d_8 defined by

$$d_8(p,q) = \texttt{max}\left(|x_p - x_q|, |y_p - y_q|\right)$$

where p and q are points of an image, (x_p, y_p) and (x_q, y_q) their coordinates.

It is also possible to define weighted approximations that fits more closely the Euclidean distance. The distance is computed as a weighted sum of a purely diagonal path defined by

$$\texttt{min}\left(|x_p - x_q|, |y_p - y_q|\right)$$

and of a vertical or horizontal path defined by

$$\texttt{max}\left(|x_p - x_q|, |y_p - y_q|\right) - \texttt{min}\left(|x_p - x_q|, |y_p - y_q|\right))$$

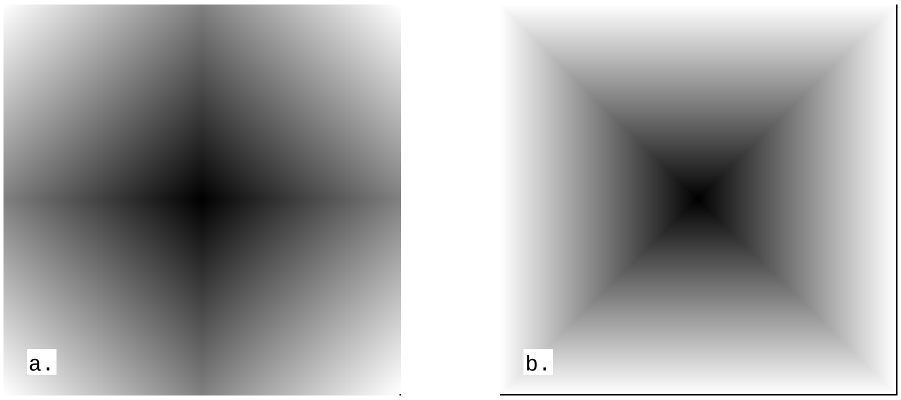

Figure 3: Distance from a pixel with d_4 and d_8 computed with ANET.

The distance from pixel p to pixel q is then:

$$d(P,Q) = a \times \texttt{min}\left(|x_p - x_q|, |y_p - y_q|\right) \quad + \\ b \times \left(\texttt{max}\left(|x_p - x_q|, |y_p - y_q|\right) - \texttt{min}\left(|x_p - x_q|, |y_p - y_q|\right)\right)$$

For instance, the values $(a,b) = (3,2)$ or $(a,b) = (7,5)$ for diagonal and non-diagonal links improve the preciseness of the distance approximation [5].

All these approximations of the Euclidean distance can be computed on the whole mesh by a single `dirassoc` primitive, instanciated with an r-operator. Indeed, computing such a distance between pixels p and q is similar to the search of a shortest path in the mesh with or without diagonal connections, and with or without weights on edges. Weights of the edges can be coded using an r-mapping like $x \mapsto x + \text{weights}$. Choosing the shortest path is done from node to node by the `min` operator. Hence distances are computed with r-operators such as $x, y \mapsto \texttt{min}(x, y + \text{weight})$. Note that when a binary operator is given, it is extended to nine incoming variables (eight neighbors plus the value of the node), using r-associativity (see § 2.4).

For the city block metric d_4, all weights are equal to one. So we use the `minc` operator introduced in § 2.4. The diagonal connections are not taken into account thanks to a specific `snet`. Consider a `pvar` D equal to the maximal value everywhere except on pixel p. Then the d_4 distance from pixel p to all other pixels is done as follows (see Figure 3-a):

```
pvar<mask> grid = 0xaa;
D = minc-dirassoc(D, grid)
```

For the chess board d_8 metric, the considered subgraph is the whole mesh, and all the weights are equal to one. We still use the `minc` operator (see figure 3-b):

```
pvar<mask> mesh = 0xff;
D = minc-dirassoc(D, mesh)
```

For the weighted approximations of the Euclidean distance, the whole mesh is considered but the weights depends on the kind of edges. The diagonal links have the weight a and all other links (horizontal or vertical) have the weight b. We then define the $\texttt{minc}_w$ r-operator that applies the r-mapping $x \mapsto x + b$ on diagonal links, and the r-mapping $x \mapsto x + a$ on others

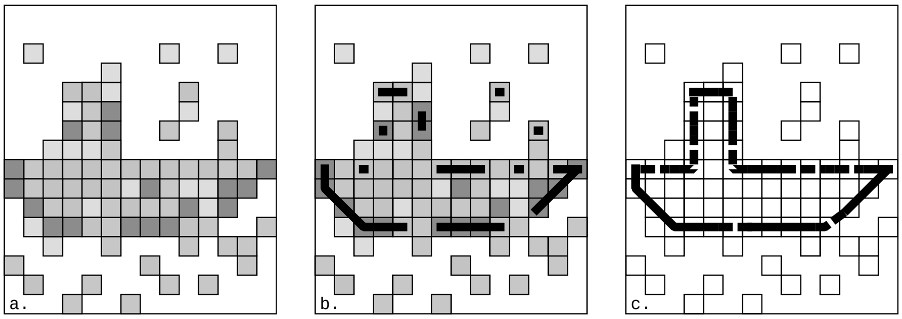

Figure 4: Contour closing: connecting parts of contours

links:

$$(x_0, x_1, \ldots, x_8) \mapsto \texttt{min}(x_0, x_1 + a, x_2 + b, x_3 + a, x_4 + b, x_5 + a, x_6 + b, x_7 + a, x_8 + b)$$

Given this operator, the computation is done as for d_8:

```
pvar<mask> mesh = 0xff;
D = minc_w-dirassoc(D, mesh)
```

3.2 Contour closing

Contour closing is a usual problem in image analysis. While low-level contour extractors can compute a measure of the intensity of a contour in any point, noise can blur the information (Figure 4-a), and it is generally required to find the proper extension that will close contours (Figure 4-b and c). This can be done by searching the best path that connects some contour extremities.

After contour extraction, points with a contour intensity above a threshold are marked as certain, and the other ones are labelled with a quantity inversely proportional to their contour intensity. The problem is to find the sequence of points whose global sum is minimal. This is the search of a shortest path on the inter-pixel graph, where the weight of an edge is equal to the local contour measure.

For implementing contour closing with ANET, portions of coutours should be first extract with a low level operator (threshold, filter...). By this way, each pixel inherits of a value in a `pvar c` that measures its ability to be part of a contour; for instance `c[p] = 0` means that the pixel `p` is part of a contour while `c[p] = 255` means that `p` may not appear at all to be in a contour. Next, each edge $(\texttt{p}, \texttt{q})$ of the inter-pixel oriented graph is assigned a weight equal to `c[q] - c[p]`, and all edges with negative weights are masked. Finally, each pixel computes its smaller weighted length to any pixel belonging to a contour, thanks to a single asynchronous $\texttt{minc}_w$-`dirassoc` (the operator $\texttt{minc}_w$ has been defined in § 3.1). Pixels which are closed to a contour are then considered as part of the contour, which has then grown. Such a process could be iterated (if necessary) until stability of the contour. Note that it is also possible to improve the method by considering contour orientation; the weight of the edge from pixel `p` to vertex `q` will be reduced if contour orientation in vertex `p` is directed towards `q` and conversely.

3.3 Voronoï segmentation

Image segmentation is a computing intensive task in image analysis. Its purpose is to gather in the same region similar pixels, with respect to some homogeneity criterion. For instance, one could try to form regions whose variance is below a certain threshold, or whose difference between maximum and minimum value is bounded.

To implement that task, a classical method is to use the so-called split-and-merge algorithm [13]. First, the image is recursively split into regions, while the homogeneity criterion is not satisfied everywhere. Then, these regions are merged until no merging can be done that satisfies the homogeneity criterion.

The associative mesh model is perfectly adapted to this scheme, since it allows to efficiently perform the global region manipulations, with `vnets` [7]. Computing a criterion on the regions is then easily done with some `dirassoc` on the `vnet`. Moreover fusions of regions can simply be done in the `vnet` level (see *e.g.* Figure 2).

Several schemes can be used for splitting the image. A classical one is to define a recursive geometrical decomposition *à la quad-tree*. This method is quite straightforward to implement, as the graph decomposition of the image can only rely on the pixels coordinates. A smarter way is to use a decomposition of the region based on the Voronoï polygons [22]. In that case, for spliting inhomogeneous regions of the image, a pixel must be choosen inside this region, either randomly or by using some heuristics. This pixel $\mathtt{p}_i$ will be the *seed* of a new Voronoï region, defined as the set S_i of pixels whose distance to $\mathtt{p}_i$ is smaller than the distance to any other pixel $\mathtt{p}_j$, $i \neq j$ [1].

Computing the Voronoï regions requires to compute the shortest length of each pixel to any seed, and then to connect pixels that have choosen the same seed. Hence, a single asynchronous `minc-dirassoc`, followed by some short operations for connecting regions, gives the Voronoï tessellation. It is important to notice that without asynchronism, the computation of the Voronoï regions requires many iterations, for improving step by step the regions until a best partition of the image is found [7]. Asynchronism leads to some local relaxation phenomena in the mesh, and accelerates the convergence rate: there is no (unecessary) synchronisation between iterations. Moreover, broadcast of values from one pixel to its entire region is quickly performed by an asynchronous `max-dirassoc`.

We present here an ANET program for Voronoï segmentation, that constructs the `snet sRegions` of the Voronoï regions from a `pvar pSeed`, such that `pSeed[i] = i` on seeds and `0` elsewhere [8]. Note that since the mesh is not a planar graph, some precautions must be taken to avoid overlapping of regions due to crossing of diagonal connections; this algorithm is shortest on planar graphs (*e.g.* 4-connected of hexagonal grid). Nevertheless, one can note the small size of the program.

```
snet sMesh(0xff);   // subnet of the whole mesh.
snet sRegion; //subnet of Voronoi regions to be constructed.
pvar<int> pSeed;    // 0 everywhere, selfaddress on seeds.

// Compute the distance of each pixel from its closest seed.
pvar<int> pDist;    // pvar of int to store the distances.
where(pSeed != 0) {
  pDist = 0;            // distance is 0 on seeds,
} elsewhere {
```

```
  pDist = MAX_INT;      // and largest integer elsewhere.
} endwhere;
pDist = minc-dirassoc(sMesh, pDist); // Distance comput.
//pDist = distance to the nearest seed.

// Next, construction of the Voronoi regions.
// 1. Construction of the DAG of the smallest ancestors.
pvar<int> pMin;
pMin = min-step-dirassoc(sMesh, pDist); // Local minima.
// connecting each pixel to its smallest neighbors:
link-with-equals(sRegions, pMin, pDist, sMesh);
where( pDist == 0 ){
  sRegions = 0; //Seeds have no ancestors.
} endwhere;
// sRegions is now a spanning DAG.

// 2. Construction of a spanning planar forest from DAG.
// Looking first at vertical or horizontal neighbors
// in the DAG to avoid crossing of diagonal links.
for (unsigned i = 0; i < 8; i+=2){
  // Scanning neighbors 0, 2, 4, 6, (ie. mask 10101010)
  where( (sRegions & Mask(1 << i)) != 0 ){
    // False on nodes already on the forest.
    // Incoming links are added to sRegions.
    sRegions = Mask(1 << i);
  } endwhere;
}
// and then looking at diagonal neighbors in the DAG.
for (unsigned i = 1; i < 8; i+=2){
  // Scanning neighbors 1, 3, 5, 7, (ie. mask 01010101)
  where( (sRegions & Mask(1 << i)) != 0 ) {
    // False on nodes already on the tree.
    // Incoming links are added to sRegions.
    sRegion = Mask(1 << i);
  } endwhere;
}
// sRegion is now a spanning planar forest.

// 3. Construction of the regions from the forest.
// (regions do not overlap thanks to planar tree).
 // Broadcast seed address over each tree of the forest.
pMin = max-assoc(sRegions, pSeed);
 // Connecting pixels belonging to the same region.
link-with-equals(sRegions, pMin, pMin, sMesh);
// sRegions is now a subgraph of the Voronoi regions.
```

Figure 5 shows a Voronoï split of the image *lena*. Note that after the `snet sRegions` has been constructed, virtual structures (see § 2.2) allow to easily work on regions level:

Figure 5: Example of Voronoï split on *lena*. A single `minc-dirassoc` allows to compute Voronoï regions from seeds.

```
vnet vRegion(sRegion); // Defining the region's graph.
with( vnet )
  ...                  // Working at the region's level
endwith;
```

3.4 Watershed segmentation

Watershed image segmentation proceeds by considering an image as a relief (see figure 6-left). Local minima are considered as water sources that flood the terrain. Small valleys are over-flooded, but large ridges stop water propagation, and prominent depressions appear as large lakes. Whenever water rises, details disappear in the image. The algorithm is iterative: every step corresponds to an increase in water height. Characteristics of temporary lakes — such as surface, depth, volume, *etc* — can be controled at each step in order to determine if it corresponds to an important enough relief and then to forbid its fusion with surrounding lakes. Water flow stop criterions are parameters of the algorithm. Figure 6-middle illustrates water rise in the left relief, while the final segmentation result is given by Figure 6-right (upper view). We indicate here an algorithm to perform watershed segmentation with the associative nets model by means of r-operators.

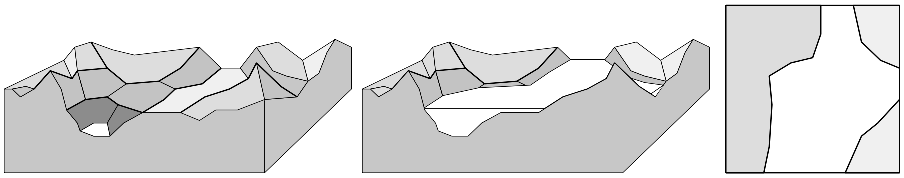

Figure 6: Watershed segmentation (principle).

The first step consists in the construction of the subnet that water will follow. It is composed of edges directed towards neighbors with an higher altitude (Figure 7).

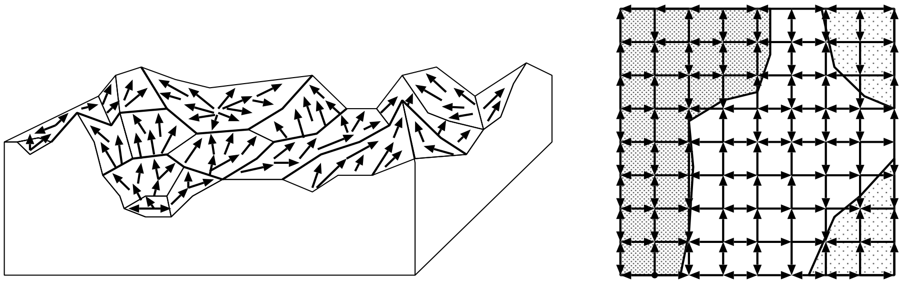

Figure 7: Watershed segmentation (algorithm).

Next, weights of these edges have to be determined, in order to control the flooding of the terrain. Such weights influence the heuristic. They must reflect a faster water progression on smoother surfaces than on abrupt ones. Two criterions must be combined: grey level difference and distance. As an example, one can affect to the edge $(\mathtt{p},\mathtt{q})$, that connect pixels `p` and `q` in the image coded by `pvar I`, the weight w defined by

$$w\,(\,(\mathtt{p},\mathtt{q})\,) = \mathtt{I[q]\ -\ I[p]} + 1$$

Note that all the weights are positive in the water flooding subnet.

The third step consists in computing the flooding on the weighted subnet, thanks to the $\mathtt{minc}_w$ r-operator introduced in § 3.1. Every pixel that is a local minimum in the image is a water source. Such a source is initialized with the corresponding pixel value. Other pixels are initialized with the maximum value (elementary element of the $\mathtt{minc}_w$ operator). Water rise is done by a single asynchronous $\mathtt{minc}_w$`-dirassoc`. As for Voronoï regions construction, the absence of synchronisation during the flooding accelerates the computation.

During the fourth step, some regions should disappeared, using a given criterion, say the height of the region. The value obtained in every node after the $\mathtt{minc}_w$`-dirassoc` is related to the height of the water and the distance to the source. Meanwhile, as *all* the regions have been over-flooded by water, the height of water in every basin must be determined, in order to decide which of them should be kept. For doing that, the region graph is inverted using the `invert` primitive: each edge direction is changed. Pixels on a ridge do not have ancestors in this graph. All the pixels are set to the maximum value (elementary element of the s-operator `min`), but the ridge pixels that keep their value. By means of a `min-dirassoc`, every source knows the height of the lower ridge of the region it has flooded. Source can then compute the height of the lake it belongs to, and decides to disappear or not, according to the minimal height parameter of the algorithm.

After some sources have been suppressed, a new iteration can start. This iterative process stops when no source have been removed. Figure 7 illustrates the final result.

Note that more complex criterions can be considered too. For computing the surface of a regions, pixels higher than the lowest ridge are temporary deconnected of the flooding subnet; the area is then simply computed with a +`-dirassoc` on a leaves-to-root oriented spanning-tree rooted on sources, provided that all the nodes have been set to 1 (as explained in § 2.4). The volume can be estimated by summing the difference between every pixel and the height of the lowest ridge.

3.5 Mathematical morphology

We give as a last example of the use or r-operators in asynchronous computations the implementation of opening and closing by reconstruction. This consists of mathematical morphology operators, that allow to suppress noise, while preserving contours. They belong to the set of morphological reconstruction filters that are widely used in the early preprocessing steps and are a special instance of *connected operators* that operate on flat zones of the image [20, 21].

The *erosion* operator assigns to the pixel the smallest pixel value belonging to a neighborhood (*structuring element*) centered on it. On the frequent case of the 3×3 structuring element, it is computed on the mesh by $\mathtt{min}(x_0, x_1 \dots, x_8)$, where x_0 is the pixel value, and $x_1, \dots, x_8$ the values ot its 8 neighbors. We denote by $\mathtt{Min}^k(\mathtt{I})$ the kth iterate of this unary (global) operator on the image $\mathtt{I}$. Similarly, the *dilation* operator assigns to the pixel the largest value in the structring element. On the 3×3 structuring elements, it is computed by $\mathtt{max}(x_0, \dots, x_8)$. We denote by $\mathtt{Max}^k(\mathtt{I})$ the kth iterate of this unary (global) operator on an image $\mathtt{I}$.

The *geodesic erosion* consists to limit the erosion process of an image $\mathtt{I}$ by computing the maximum of the eroded image $\mathtt{Min}^1(\mathtt{I})$ and of a reference image $\mathtt{R}$: $\mathtt{Geoder}(\mathtt{I},\mathtt{R}) = \mathtt{max}(\mathtt{Min}^1(\mathtt{I}),\mathtt{R})$. We states that $\mathtt{Geoder}^0(\mathtt{I},\mathtt{R}) = \mathtt{I}$, and $\mathtt{Geoder}^k(\mathtt{I},\mathtt{R}) = \mathtt{Geoder}(\mathtt{Geoder}^{k-1}(\mathtt{I},\mathtt{R}),\mathtt{R})$. Similarly, the *geodesic dilation* is defined by: $\mathtt{Geodil}(\mathtt{I},\mathtt{R}) = \mathtt{min}(\mathtt{Max}^1(\mathtt{I}),\mathtt{R})$. We states that $\mathtt{Geodil}^0(\mathtt{I},\mathtt{R}) = \mathtt{I}$, and $\mathtt{Geodil}^k(\mathtt{I},\mathtt{R}) = \mathtt{Geodil}(\mathtt{Geodil}^{k-1}(\mathtt{I},\mathtt{R}),\mathtt{R})$.

Finally, a *closing by reconstruction* consists to apply until stability — in a synchronous way — a geodesic erosion on a k-times dilated image $\mathtt{Max}^k(\mathtt{I})$, with the initial image as reference: $\mathtt{Closerec}_k(\mathtt{I}) = \mathtt{Geoder}^\infty(\mathtt{Max}^k(\mathtt{I}),\mathtt{I})$. Similarly, the *opening by reconstruction* consists to apply successive geodesic dilatations on a k-times eroded image $\mathtt{Min}^k(\mathtt{I})$, taking the initial image as reference: $\mathtt{Openrec}_k(\mathtt{I}) = \mathtt{Geodil}^\infty(\mathtt{Min}^k(\mathtt{I}),\mathtt{I})$.

Closing and opening by reconstruction are actually filters used to simplify the images. These global operators need several (synchronized) iterations on the whole image before converging. In [19], the authors proposed to extend asynchronous relaxation — first defined for matrix computation (see *e.g.* [23]) — to the computation of these operators. They show that asynchronous iterations lead to the same result as synchronous ones, and they observed that parallel asynchronous iterations take nearly twice less iterations than synchronous one.

One can prove that opening and closing by reconstruction can be locally performed by iteratively apply the following operators (x_0' denotes the pixel value in the initial image taken as reference):

$$\begin{aligned} \mathtt{closerec}\,(x_0, \dots, x_8) &= \mathtt{min}\,(x_0, \mathtt{max}\,(x_0', x_1), \dots, \mathtt{max}\,(x_0', x_8)) \\ \mathtt{openrec}\,(x_0, \dots, x_8) &= \mathtt{max}\,(x_0, \mathtt{min}\,(x_0', x_1), \dots, \mathtt{min}\,(x_0', x_8)) \end{aligned}$$

These operators are idempotent r-operators, and then any `dirassoc` instanciated by them will converge. This means that a single asynchronous `dirassoc` allows to compute either opening or closing by reconstruction.

4 Conclusion

The associative nets constitute a well suited programming model for image analysis. It is based on the data-parallelism paradigm, and provides specific data structures and computing primitive. It has been designed either for hardware or software parallel implementations.

This model allow to replace some synchronous computations by asynchronous ones. Such asynchronous computations have many advantages for implementation, both in hardware or software. But they also provide expressiveness for programming. As an example, a single `dirassoc` can replace many synchronous iterations.

However, not all the computations can be replaced by asynchronous one, depending on the operator used. Thanks to new operators — called r-operators — that encompass the class of previous authorized operators, many algorithms can now take benefit of asynchronous computation.

This paper was dedicated to the interest of such r-operators and of parallel asynchronous computations in image analysis. We showed that many fundamental image analysis problems can be programmed with asynchronous computations. A single asynchronous computation allow to (i) compute a distance transform, (ii) search a best path for closing contours, (iii) compute Voronothe ï regions in a Voronoï segmentation, (iv) simulate the water flooding in a watershed segmentation, and (v) compute an opening or a closing by reconstruction.

These algorithms illustrate the interest of the r-operators, and of the asynchronous computations for image analysis. Asynchronism was used for implementation purpose. We think that it can *also* be used with benefit at the programming level.

References

[1] Ahuja, N., An, B., and Shachte, B. Image representation using voronoi tesselation. *CVGIP 29* (1985), 286–295.

[2] Baudet, G. M. Asynchronous iterative methods for multiprocessors. *Journal of the ACM 25*, 2 (April 1978), 226–244.

[3] Biancardi, A., and Mérigot, A. Connected component support for image analysis programs. *Proc. 13th ICPR, IEEE Press D* (1996), 620–624.

[4] Blelloch, E. G. *Vector Models for Data-Parallel Computing.* MIT Press, Cambridge, Ma., USA, 1990.

[5] Borgefors, G. Distance transformations in digital images. *CVGIP 34* (1986), 344–371.

[6] Ducourthial, B. New operators for computing with *associative nets.* In *Proceedings of the Fifth International Colloquium on Structural Information and Communication Complexity (SIROCCO'98), Amalfi, Italia* (June 1998), Carleton Scientific, pp. 55–65.

[7] Ducourthial, B., Constantinescu, G., and Mérigot, A. Implementing image analysis with a graph based parallel computing model. *Computing Supplement 12* (1998).

[8] Ducourthial, B. *Les réseaux associatifs, un modèle de programmation à parallélisme de données, pour algorithmes et données irréguliers, à primitives de calcul asynchrones.* PhD thesis, IEF, Université Paris-Sud, France, January 1999.

[9] Ducourthial, B., Mérigot, A. and Sicard, N. Anet: a programming environment for parallel image analysis. In *Proc. of the 2000 IEEE International Workshop on Computer Architectures for Machine Perception (CAMP 2000)* (Padova, Italia, Sept. 2000), pp. 280–289.

[10] Dulac, D., Mohammadi, S., and Mérigot, A. Implementation and evaluation of a parallel architecture using asynchronous communications. In *Proc. of the 1995 International Workshop on Computer Architecture for Machine Perception (CAMP'95)* (Pavia, Italia, Sept. 1995), V. Cantoni, L. Lombardi, et al., Eds., IEEE Press, pp. 106–111.

[11] Dulac, D. *Contribution au Parallélisme Massif en Analyse d'Images : Une Architecture SIMD Fondée sur la Reconfigurabilité et l'Asynchronisme.* PhD thesis, IEF, Université Paris-Sud, France, January 1996.

[12] Fountain, T. J. *Processors Arrays: Architecture and Applications.* Academic Press, 1987.

[13] Horowitz, S., and Pavlidis, T. Picture segmentation by a tree traversal algorithm. *Journal of the ACM 23*, 2 (1976), 368–388.

[14] Mérigot, A. Associative nets: A graph-based parallel computing net. *IEEE Transactions on Computers 46*, 5 (May 1997), 558–571.

[15] Nacken, P. *Image Analysis Methods Based on Hierarchies of Graphs and Multi-Scale Mathematical Morphology.* PhD Thesis, University of Amsterdam, Jun. 1994.

[16] Owens, R. M., Irwin, M. J., Nagendra, C., and Bajwa, R. S. Computer vision on MGAP. In *Proc. of the 1993 Workshop on Computer Architecture for Machine Perception* (New Orleans, Louisiane, USA, Dec. 1993), L.S. Davis, M.A. Bayoumi, and K.P. Valavanis, Eds., pp. 337–341.

[17] Privat, G., Robin, F., Renaudin, M., and EL Hassan, B. A fine-grain asynchronous VLSI cellular array processor architecture. In *Proceedings of the IEEE International Symposium on Circuits and Systems* (Seattle, USA, April 1995), pp. 1041–1044.

[18] Robin, F. *Etude d'architectures VLSI numériques parallèles asynchrones pour la mise en œuvre de nouveaux algorithmes d'analyse et rendu d'images.* PhD thesis, ENST, Grenoble, France, Octobre 1997.

[19] Robin, F., Privat, G. and Renaudin, M. Asynchronous relaxation of morphological operators: a joint algorithm-architecture perspective. *International Journal of Pattern Recognition and Artificial Intelligence 11*, 7 (1997), 1085–1094.

[20] Salembier, P. and Pardàs, M., Hierarchical morphological segmentation for image sequence coding. *IEEE Transactions on Image Processing 3*, 5 (1994), 639–651.

[21] Salembier, P. and Serra, J., Flat zones filtering, connected operators and filters by reconstruction. *IEEE Transactions on Image Processing 4*, 8 (1995), 1153–1160.

[22] Tuceryan, M., and Jain, A. Texture segmentation using Voronoi polygons. *IEEE Transactions on PAMI 13* (1990), 211–216.

[23] Üresin, A., and Dubois, M. Sufficient conditions for the convergence of asynchronous iterations. *Parallel Computing 10* (1989), 83–92.

[24] Üresin, A., and Dubois, M. Effects of asynchronism on the convergence rate of iterative algorithms. *JPDC 34* (1996), 66–81.