On the study of VLSI derivation for optical flow estimation

Étienne Mémin and Tanguy Risset
IRISA/INRIA 35042 Rennes Cedex, France; risset@irisa.fr

Abstract

In this paper we propose studying several ways to implement a realistic and efficient VLSI design for a gradient based dense motion estimator. The kind of estimator we focus on belongs to the class of differential methods. It is classically based on the optical flow constraint equation in association with a smoothness regularization term and also incorporates robust cost functions to alleviate the influence of large residuals. This estimator is expressed as the minimization of a global energy function defined within the framework of an incremental formulation associated to a multiresolution setup. In order to make possible the conception of efficient hardware, we consider a modified minimization strategy. This new minimization strategy is not only well suited to VLSI derivation, it is also very efficient in terms of quality of the result. The complete VLSI derivation is realized using high level specifications.

1 Introduction

Optical flow estimation is one of the oldest reconstruction problems. This problem which consists of computing the 2D apparent motion field between two consecutive frames of an image sequence, is indeed an inescapable prerequisite in a wide range of applications such as passive navigation, stereovision, image sequence restoration, video compression, etc. Several techniques have been proposed to estimate velocity fields. Among others, one can distinguish spatio-temporal filtering methods [13], tensor-based techniques [20], correlation based techniques [1] or minimizing energy based techniques [18]. The latter, despite leading to intensive calculations, yields the most accurate flow fields in terms of precision [21] or in terms of spatial discontinuity estimation [5, 23]. Nevertheless, the ability of such methods to be embedded in real time applications is limited by the amount of computing power needing to solve them. The computation cost is even so high that nowadays it is unimaginable to reach that goal on conventional computers. Only dedicated architectures have the capability to respond to such throughput demands. The purpose of the paper is to investigate of a hardware implementation of a dense optical flow estimator. Such study does not pretend to propose a commercial product but rather aims at describing several approaches to an efficient, realistic and feasible hardware design for dense motion estimation.

In that context, the kind of motion estimator we focus on belongs to the class of differential methods. Those methods are expressed as the minimization of a global cost function. The objective function, usually composed of two terms, is build on the standard basis of the brightness constancy assumption in association with a regularizing smoothness *a priori*.

The first term relies on an infinitesimal expansion of the brightness constancy known as the *optical flow constraint equation* (OFCE). This equation links the projection of the unknown velocity fields on the spatial gradient of the luminance function to the temporal partial derivative of the luminance function.

The second term, the smoothness term, allows tackling the ill posed nature of the problem at hand in enforcing piecewise continuous solutions.

A standard quadratic energy modeling method suffers from well known weaknesses: the OFCE does not hold when spatial or temporal variations of the luminance function are too large; and also, the boundaries which demarcate the different apparent motions coexisting within the same scene are ignored by a "blind" *a priori* smoothing, which results in a bad estimation nearby these border lines. Though different in nature, these two problems can be efficiently attenuated and isolated considering non-quadratic norms such as those introduced in the field of robust statistics [19]. Nevertheless, the introduction of robust estimators in energy-based image applications leads most of the time to a global non-linear minimization in presence of numerous local minima. Even within an *incremental multiresolution* formulation of the problem (which is almost inescapable in case of long range motions to be estimated), one has to deal with a sequence of global optimization problems which remain tricky. However, thanks to a reformulation result [8, 15] the associated minimization can be carried out, using new auxiliary variables, as a sequence of weighted quadratic problems. The overall optimization consists in that context of considering alternative minimizations with respect to the new variables (or weights) and with respect to the unknowns.

A hardware derivation of such an incremental robust optical flow estimator is far from being straightforward. It posseses some inherent "algorithmic limitations" that must be adressed. First of all, this kind of estimator is usually defined within a multiresolution setup. This framework involving a pyramidal decomposition of the images is combined with an incremental formulation of the optical flow estimation problem. Such a formulation is similar in spirit to *Gauss-Newton* minimization and is insepareable of a non regular registration phase very difficult to parallelize efficiently. This registration usually occurs between two resolution levels. In this paper, we propose performing the registration periodically during the estimation of the current level. In addition, the associated alternate minimization necessitates the successive use of an iterative minimization method. In our context, the particular choice of a method must be done very carefully. It should exhibit a good trade off between convergence efficiency and the feasibility of a parallel implementation.

A VLSI implementation is also constrained by some severe "hardware limitations" such as fixed point arithmetic precision and input/output bandwidth. These constraints may indeed completely change the behavior of an initial estimator built assuming large floating point precision. Such a two-step derivation methodology (algorithmic specification and hardware derivation) are tightly interwoven. Whenever the concerned chip does not satisfy some prescribed constraint (execution time cycle, or surface area), the definition of the algorithm or the design choices must be further modified. A reliable tool to automatically (or semi-automatically) derive specialized hardware from algorithmic specifications attenuates these difficulties. It not only greatly facilitates the VLSI design, but also allows the redefinition of the algorithm. Any change on the algorithm may easily be tanslated to the hardware with the VLSI derivation tool.

Following these general remarks, the purpose of this paper is to show that high level design methodologies can successfully be used to design efficient dedicated hardware for a "realistic" optical flow estimator.

The outline of the paper is the following. First, we present the incremental multiresolution motion estimator. We will also show how interaction with the VLSI design has led us to consider a modified incremental non linear minimization strategy. The resulting minimization algorithm is not only well suited to VLSI implementation, but also has proved to be very efficient. This efficiency is demonstrated experimentally. The last part of the paper is devoted to presenting a high level design methodology.

2 Robust incremental optical flow estimation

Optical flow estimation aims at recovering the apparent displacement field $\boldsymbol{w} = \{\boldsymbol{w}_s, s \in S\}$ between two consecutive frames in an image sequence. Differential motion estimator are based on the so called optical-flow constraint equation (OFCE)[18]:

$$\boldsymbol{\nabla} f(s,t)^T \boldsymbol{w}_s + f_t(s), = 0$$

where $\boldsymbol{\nabla} f$ stands for the spatial gradient of the luminance function f and $f_t(s) \triangleq f(s,t+1) - f(s,t)$ denotes the luminance variation at location s. This equation results from a linearization of the brightness constancy assumption $f(s+\boldsymbol{w}_s, t+1) - f(s,t) = 0$.

As a Taylor expansion, the OFCE assumes that the unknown displacement remains in the domain of linearity of the luminance function. This hypothesis is particularly weak for long range displacement or around sharp edges. To circumvent these limitations an incremental version of this equation is usually considered. This technique, which may be related to non-linear least squares *Gauss-Newton* method [4, 23], is generally used in combination with a standard multiresolution setup [5, 12]. Such a framework, involving a pyramidal decomposition of the image data, is unavoidable in numerous situations. In the following, we shall assume to work at a given resolution of such pyramidal multiresolution structure. However, one has to keep in mind that the expressions and computations are meant to be reproduced at each resolution level according to a coarse to fine strategy.

Let us now suppose that a rough estimate $\boldsymbol{w} = \{\boldsymbol{w}_s, s \in S\}$ of the unknown velocity field is available (e.g., from an estimation at lower resolution or from a previous estimation). Under the constant brightness assumption from time t to $t+1$, a small *increment field* $\boldsymbol{dw} \in \Omega \subset (\mathbb{R} \times \mathbb{R})^S$ can be estimated by minimizing the functional $\mathcal{H} \triangleq \mathcal{H}_1 + \alpha \mathcal{H}_2$, with [5, 23]:

$$H_1(\boldsymbol{dw}; f) \triangleq \sum_{s \in S} \rho_1[\boldsymbol{\nabla} f(s+\boldsymbol{w}_s, t+1)^T \boldsymbol{dw}_s + f_t(s,t,\boldsymbol{w}_s)] \tag{1}$$

$$H_2(\boldsymbol{dw}; \boldsymbol{w}) \triangleq \sum_{<s,r> \in \mathcal{C}} \rho_2\left[\|(\boldsymbol{w}_s + \boldsymbol{dw}_s) - (\boldsymbol{w}_r + \boldsymbol{dw}_r)\|\right] \tag{2}$$

where $\alpha > 0$ is a parameter balancing the two energy terms, $\mathcal{C}$ is the set of neighboring site pairs lying on grid S equipped with some neighborhood system ν and functions ρ_1 and ρ_2 which are standard *robust M-estimators* (with parameters σ_1 and σ_2). Functions ρ_1 and ρ_2 penalize the *deviations* from the data model (i.e., the OFCE in (1)) and from the smoothing prior (2).

Unlike a quadratic penalty, these robust functions which are often non-convex for improved robustness [10] enable us reducing greatly the contribution of large residuals. Furthermore, assuming the concavity of $\phi(v) \triangleq \rho(\sqrt{v})$) [8, 15], these functions possess a nice property allowing them to handle easily the associated non-linear minimization. Indeed, any multidimensional minimization problem of the form "find $\arg\min_x \sum_i \rho[g_i(x)]$" can be turned into a dual minimization problem "find $\arg\min_{x,z} \sum_i [\tau z_i g_i(x)^2 + \psi(z_i)]$" involving *auxiliary variables* (or *weights*) z_is lying continuously in $(0,1]$ and where ψ is a continuously differentiable function, depending on ρ, and $\tau \triangleq \lim_{v \to 0+} \phi'(v)$.

The new associated minimization is then guided *alternatively* with respect to x and to the z_is. If g_is are affine, and since the expression is quadratic w.r.t. x, the corresponding minimization leads to the resolution of a sparse linear system. Due to its huge dimension, iterative methods such as conjugate gradient, Gauss-Seidel or Jacobi methods are usually chosen to solve it. However here, since x is frozen, the best weights have the following closed form [6, 8, 15]:

$$\hat{z}_i(x) = \frac{\rho'[g_i(x)]}{2\tau g_i(x)} = \frac{1}{\tau}\phi'[g_i(x)^2] \tag{3}$$

The whole alternate procedure constitutes an *iteratively re-weighted least squares* estimation (IRLS) [17]. The convergence proof of this technique toward a unique minimum (assuming the convexity of $\mathcal{H}(\boldsymbol{w})$) may be found in [8, 19]. This convergence result has been extended in [10] for the case of non-convex robust functions. In [24, 25], the convergence of this algorithms has been also demonstrated even if the convergence of the successive quadratic minimizations is not reached. In other words, it has been shown that, as far as the algorithm decreases the energy function (and if the iterates belong to a compact set), one need to do one (or few) steps of an iterative solver between two updating weights. This has led us to the concept of *full alternate minimization*. In [25] this kind of algorithm associated with Jacobi iteration[1] has been shown to be well suited to hardware derivation. The resulting algorithm is *regular* in time and does not need a complex control process to decide whether the weights have to be updated or not. Furthermore this minimization has been shown to be very efficient since it turns out to be faster than the corresponding quadratic problem (without any robust functions) in case of an image restoration application and of the same order of magnitude for a basic optical flow estimator [25].

Let us now return to our problem. The weights under consideration are of two natures: (a) *data outliers weights* (related to the dual formulation of H_1), and (b) *discontinuity weights* lying on the dual grid of S (provided by the dual formulation of H_2). The first set of weights, denoted by $\delta = \{\delta_s, s \in S\}$, allows attenuating the effect of data for which the OFCE is violated. The second one, denoted by $\beta = \{\beta_{sr}, <s,r> \in \mathcal{C}\}$, prevents over-smoothing in locations obviously exhibiting significant velocity discontinuities. The estimation is now expressed as the global minimization of $\mathcal{H}^* \triangleq \mathcal{H}_1^* + \alpha\mathcal{H}_2^*$ where:

$$\mathcal{H}_1^*(\boldsymbol{dw},\boldsymbol{\delta};f,\boldsymbol{w}) = \sum_{s\in S}\Big[\tau_1\delta_s\left[\boldsymbol{\nabla} f(s+\boldsymbol{w}_s,t+1)^T\boldsymbol{dw}_s + f_t(s,t,\boldsymbol{w}_s)\right]^2 + \psi_1(\delta_s)\Big] \tag{4}$$

$$\mathcal{H}_2^*(\boldsymbol{dw},\boldsymbol{\beta};\boldsymbol{w}) = \sum_{<s,r>\in\mathcal{C}}\left[\tau_2\beta_{sr}\|(\boldsymbol{w}_s+\boldsymbol{dw}_s)-(\boldsymbol{w}_r+\boldsymbol{dw}_r)\|^2+\psi_2(\beta_{sr})\right] \tag{5}$$

The overall Full Alternate Jacobi Minimization (FAJM (full alternate minimization based on Jacobi iteration) may be described by the following coupled recurrent equations:

$$\left\{\begin{array}{l}
\boldsymbol{\beta}^{k+1} = \arg\min_{\boldsymbol{\beta}}(\mathcal{H}^*(\boldsymbol{dw}^{k+1},\boldsymbol{\beta},\boldsymbol{\delta})) = [\ldots\frac{1}{\tau_1}\phi'(\|\boldsymbol{w}_s + \boldsymbol{dw}_s^{k+1} - \boldsymbol{w}_r - \boldsymbol{dw}_r^{k+1}\|^2)\ldots]^T. \\
\boldsymbol{\delta}^{k+1} = \arg\min_{\boldsymbol{\delta}}(\mathcal{H}^*(\boldsymbol{dw}^{k+1},\boldsymbol{\beta},\boldsymbol{\delta})) = [\ldots\frac{1}{\tau 2}\phi_2'([\boldsymbol{\nabla} f(s+\boldsymbol{w}_s,t+1)^T\boldsymbol{dw}_s^{k+1}+f_t]^2)\ldots]^T. \\
\boldsymbol{dw}^{k+1} = \bar{\boldsymbol{w}}^k + \dfrac{\delta_s[\nabla\widetilde{f}(s,t+1)\bar{\boldsymbol{w}} + \widetilde{f}_t(s)]\nabla\widetilde{f}(s,t+1)}{\gamma+\delta_s\|\nabla\widetilde{f}(s,t+1)\|^2} \text{ with } \bar{\boldsymbol{w}} = \sum_{r\in\nu(s)}\beta_{sr}(\boldsymbol{w}_r+\boldsymbol{dw}_r^k-\boldsymbol{w}_s), \\
\widetilde{f}(s,t+1) = f(s+\boldsymbol{w}^k,t+1),\ \widetilde{f}_t(s) = \widetilde{f}(s,t+1) - f(s,t) \text{ and } \gamma = \alpha\sum_{r\in\nu(s)}\beta_{sr}^k
\end{array}\right. \tag{6}$$

Due to the definition of the backward registered image, $\widetilde{f}$, the above system exhibits some bad properties from the point of view of an hardware design task. The underlying dependencies issuing from this

[1] $\mathbf{x}^{k+1} = \mathbf{x}^k + D_{\boldsymbol{z}}^{-1}(\mathbf{b} - A_{\boldsymbol{z}}\mathbf{x}^k)$ where $D_{\boldsymbol{z}}$ is the diagonal part of the system matrix $A_{\boldsymbol{z}}$ associated to weights $\mathbf{z}$, $\boldsymbol{x}$ the unknown and $\mathbf{b}$ the second member.

registration yield a *non regular* and *non local* parallel scheme. This is indeed the worst case in parallel implementation. The non-regularity is inherent in the definition of the previously estimated velocity field $\boldsymbol{w}$ and cannot be avoided. Nevertheless, the non-locality can be alleviated by considering an *incremental registration*, to handle only successive local communications.

This incremental registration consists of passing successively register image $\widetilde{f}$ from neighbor to neighbor. This may be governed by the following equations:

$$\begin{cases} \boldsymbol{x} = [\cdots [if\ |\boldsymbol{dw}_l^k| > 1\ then\ sign(\boldsymbol{dw}_l^k)\ else\ \boldsymbol{dw}_l^k]_{l=1,2} \cdots]^T \\ \boldsymbol{dw}^k = \boldsymbol{dw}^k - \boldsymbol{x} \\ \boldsymbol{w}^k = \boldsymbol{w}^k + \boldsymbol{x} \\ \widetilde{f}(s,t+1) = f(s+\boldsymbol{w}^k, t+1),\ \widetilde{f}_t(s) = \widetilde{f}(s,t+1) - f(s,t) \end{cases} \tag{7}$$

This system of equations provides an incremental warping of the interger part of the motion. It takes place at each iteration of the minimization or only once at the beginning of the estimation at the resolution level. In that case, the incremental registration has to be performed iteratively until the shift vector $\boldsymbol{x}$ become zero. The final registration, corresponding to the remaining real part of the motion, is realized by a standard bilinear interpolation.

Considering incremental registration of data at each iteration has some advantages from the hardware point of view. The associated data shifts, despite being always irregular in space, necessitates only neighbor to neighbor communication and are now incorporated into the unknown updating rule system. It provides indeed a regular behavior (in time) of the algorithm and as a consequence, no problematic dynamic control is needed to handle such registration.

Before focusing on hardware specifications, an experimental evaluation which aims at comparing the different methods presented so far is now provided.

2.1 Experimental study

The experiments have been carried out on both synthetic and real world sequences. First of all, two synthetic benchmarks have been built considering a set of forty 64×64 images of various nature (Fig. 1). On these frames, two synthetic motion fields have been applied. The first one concerns displacement of moderate magnitude (less than 2.5 pixels) whereas the second one is composed of larger displacements (magnitude from 2 to 7 pixels). Both are a globally translational velocity field perturbed on the neighborhood of high intensity gradient. Figure 1 presents some of the velocity fields of the benchmark and the images on which they have been applied.

For these two kinds of synthetic sequences, we have compared, for the incremental robust motion estimation model associated with multi-resolution strategy, three different minimization strategies. The first one is defined on the bases of a standard Iterated Re-weighted Least Squares (IRLS) associated to Jacobi iterative method. The second uses the Full Alternate Jacobi minimization (FAJM). In both of these two methods the registration is triggered at the beginning of the estimation process at each resolution level. The last method we are evaluating here consists to use FAJM method in association with an incremental registration at each iteration. Let us refer to this method by the acronym: FAJM_{inc}.

The experiments have been run with the cauchy estimator ($\rho(u) = \ln(1 + \frac{u^2}{\sigma})$) for the data term and the smoothness term. For all the methods, the same convergence criterion has been used. The convergence is considered as being reached if, during one Jacobi iteration less than 3 per cent of the

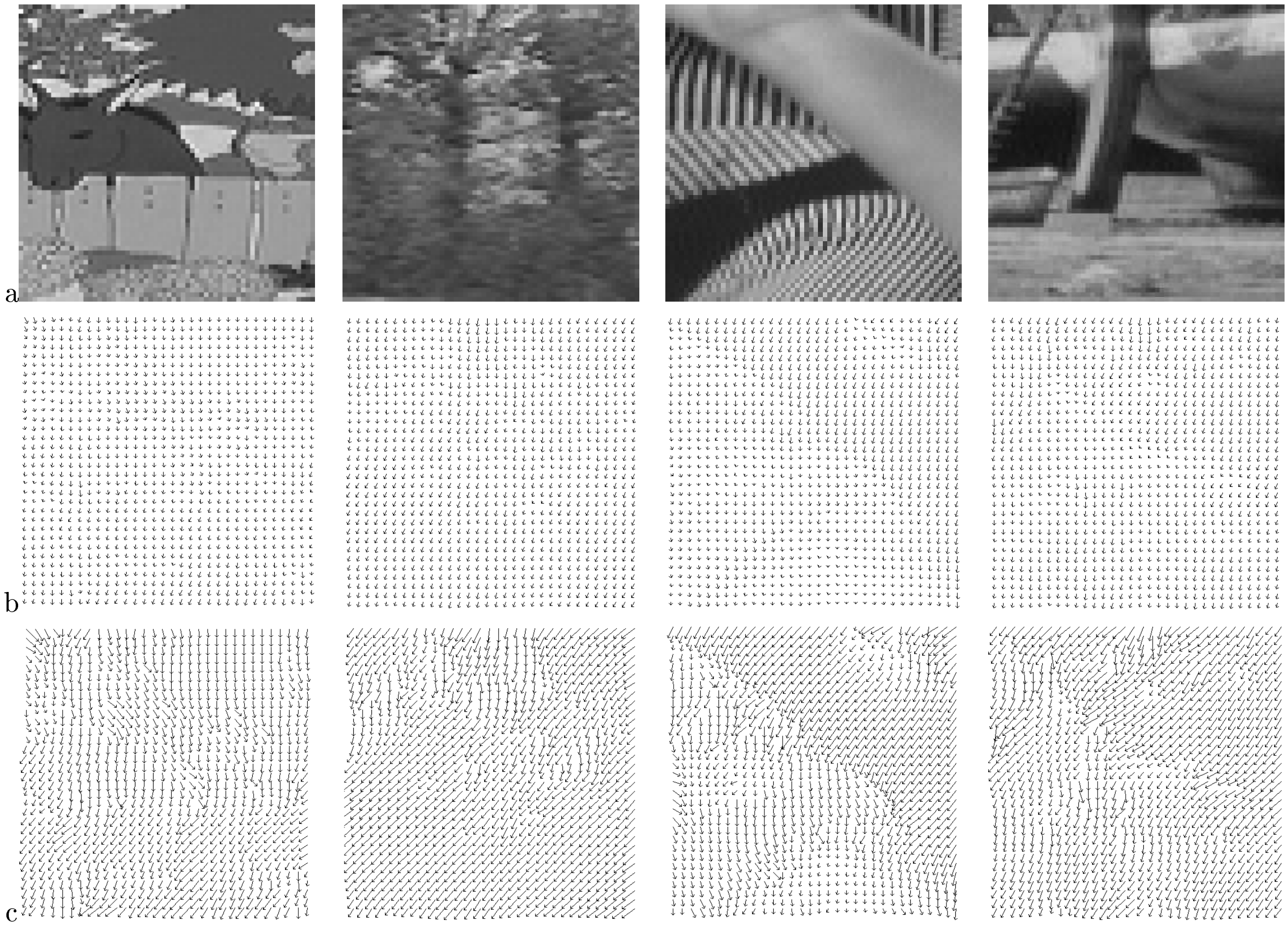

Figure 1: *Sample of the 40 images benchmark.* (a) *original images,* (b) *small displacement field,* (c) *large displacement field.*

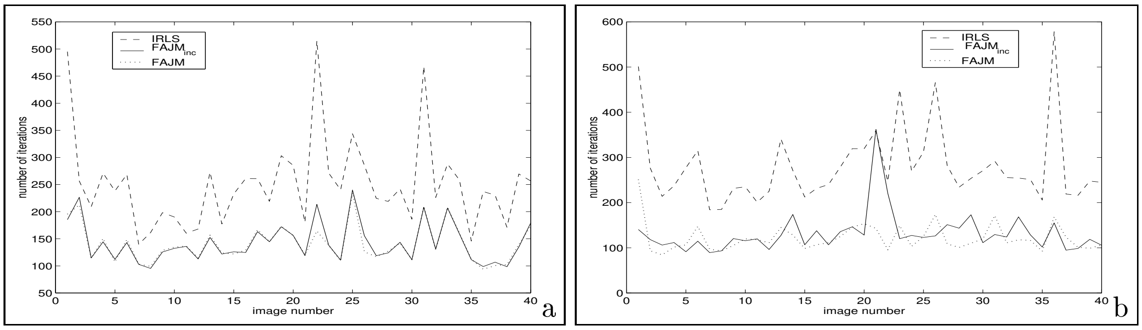

Figure 2: *Number of iterations at convergence:* (a) *small displacements;* (b) *large displacements* .

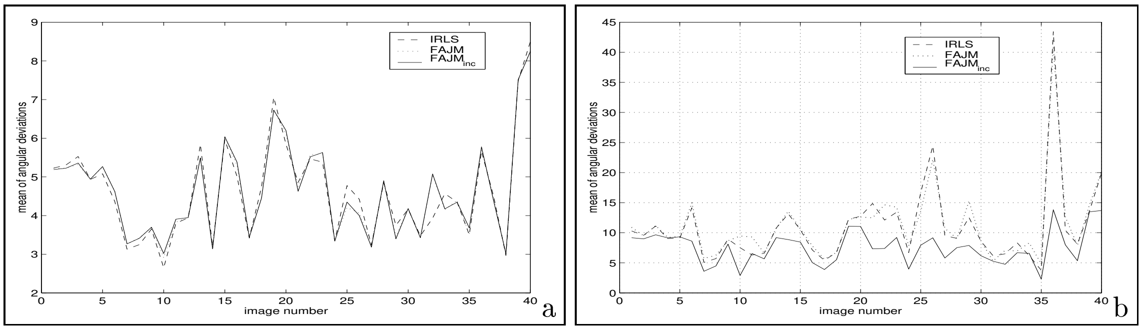

Figure 3: *Average angular discrepancies:* (a) *small displacements;* (b) *large displacements.*

image points are changed. The value of image point, s, is declared as unchanged at iteration, k, if the relative norm $\frac{\|\boldsymbol{dw}_s^k - \boldsymbol{dw}_s^{k-1}\|}{\|\boldsymbol{dw}_s^{k-1}\|}$ becomes lower than 0.01. As a result, in the IRLS case, the method converges if the linear resolution (i.e. at fixed weights) takes only one Jacobi iteration.

The first figure (Fig.2) compares, for the two kinds of motion fields, the number of iteration needed to reach convergence. As may be seen the full alternate methods (FAJM and FAJM$_{inc}$) are always faster than IRLS. The number of iterations are significantly lowered in both case. Let us note that this difference is accentuated in the case of large displacements.

The quality of convergence may be appreciated in figure 3. Following, Barron *et. al.* [3] we have plotted for all the sequences the average of the angular discrepancies between the actual flow field and the estimated one. As may be observed the results are nearly similar for both methods and none of the methods has an advantage over another. Let us remark that the final energy reached by the different methods are not comparable since, according to the registration, the phase data may change differently.

The three methods have also been run on the well known "Yosemite" synthetic sequence. The sequence is indeed one of the most difficult sequences for which a known answer exists. Table 1 summarized some results (in terms of mean and standard deviation of the angular discrepancies) obtained by the methods presented here and others proposed in the literature (see references therein). Despite being acceptable, the methods studied here have slightly worse performance than the other methods presented table 1. Nevertheless, the minimizations used in these works are far more complex and difficult to implement on hardware. For information, Black [5] uses a kind of Graduated non convexity method[7] consisting of slowly lowering the parameter associated to the robust functions; Mémin & Pérez uses a multigrid strategy [23] whereas Lai & Vemuri method is based on a preconditioned conjugate gradient solver. Futhermore, the actual Yosemite motion field is very smooth and favors IRLS method.

Finally, IRLS and full alternate methods have been run on real images. As an example figure 4 shows

Technique	Average error	Standard deviation
FAJM	4.87^o	4.35^o
FAJM$_{inc}$	4.85^o	4.33^o
IRLS	4.24^o	3.42^o
Black [5]	3.52^o	3.25^o
Mémin & Pérez [23]	2.34^o	1.45^o
Lai & Vemuri [21]	1.99^o	1.41^o

Table 1: *Comparative results computed on the* Yosemite *sequence without the sky*

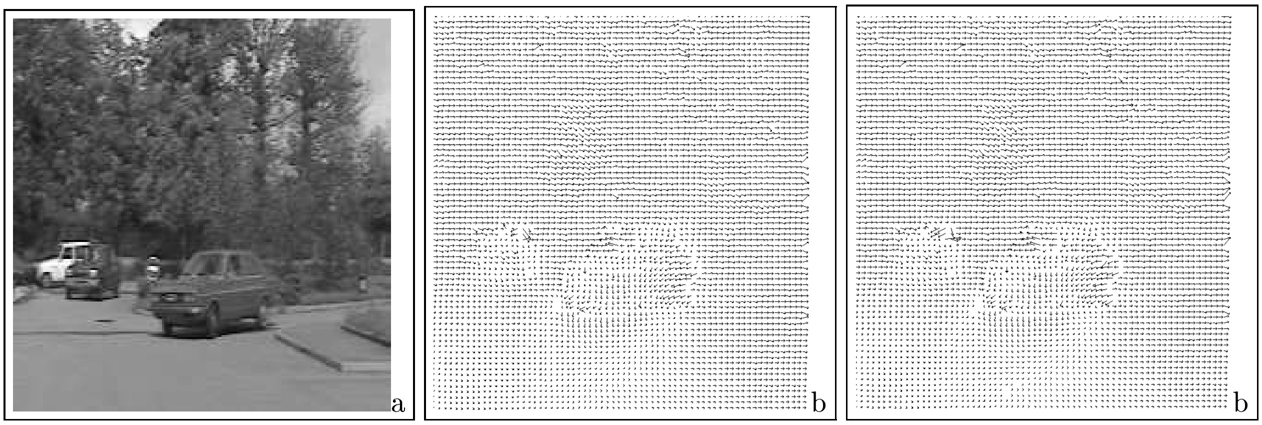

Figure 4: *Results on* Parking lot*; (*a*) first image of the sequence; (*b*) motion field computed with* IRLS*; (*c*) motion field computed with* FAJM$_{inc}$

two fields computed with the IRLS and FAJM$_{inc}$ methods on a parking lot sequence. As may be observed, the two computed fields are very nearly similar. Visually both motion fields seem to be of good quality.

As a conclusion to this study, the FAJM$_{inc}$ method provides good results from the quality point of view (this is particularly true on the benchmark defined with large displacements) and converges in fewer iterations than IRLS. In the next section, we will show why it is also well suited for hardware implementation of the multi-resolution strategy.

3 Derivation of hardware

In this section, we detail how the synthesis of dedicated hardware can be done from high level specification of algorithms.

3.1 Methodology

Hardware specification is very different from algorithm specification. First, the scientific background of the hardware synthesis community and image processing algorithm designers are generally very different, hence communication between these two communities is difficult, whenever it happens. Secondly, slight modifications of an original algorithmic specification may or may not imply very important modifications on hardware, increasing or decreasing the complexity of the circuit generated. Finally, since these two specifications are expressed in very different languages (like for instance, VHDL and C), backward annotation, from a non-satisfactory hardware, to the original algorithm is complex, and it is almost

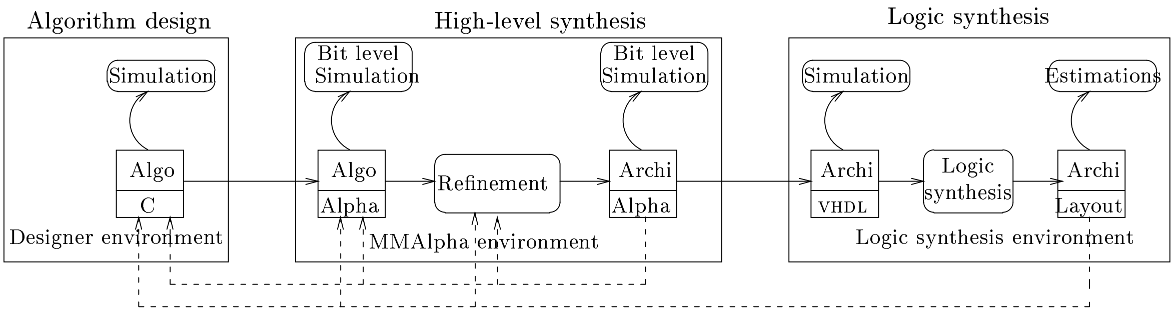

Figure 5: *The synopsis of a global design. The languages of the different specifications are indicated. Dashed lines represent possible backward annotations of specifications from lower level performance estimation.*

impossible to manage automatically (or at least semi-automatically) such backtracks. Furthermore, the inherent translation between different formats is a potential source of errors which are difficult to check.

This leads most of the time to the following strategy: definitively fixing the algorithm and then, deriving one single kind of hardware. All these difficulties increase the cost and the time to market of a dedicated hardware. Ideally, the design process should be much more flexible. To that end and in order to circumvent the aforementioned design problems, we propose here an approach based on a high level synthesis.

More precisely, our derivation process is based on the Alpha language [31, 22]. Alpha is a functional language used in a program transformation environment called MMAlpha. Manipulating algorithms in Alpha has the following advantages:

- It can describe high level functional specifications of algorithms as well as low level descriptions of hardware. Hence it provides a common useful platform for image processing engineers and hardware synthesis engineers. This is particularly convenient for bit-level simulations.

- The translation from the original high level specification into the final hardware specification is done by successive refinements of the original specification. The Alpha environment provides a set of transformations which are proved to be correct (thus no *hand made* transformation is necessary). The language allows strong static checks at each stage. The successive specifications may also be simulated at each stage.

- Once the basic design flow has been set for a particular algorithm, it can very easily be modified in order to explore the different possibilities in the design space. The original specification can also be modified according to the results of the design process.

Figure 5 represents the overall architecture of a complete design flow with a tool like MMAlpha. The design methodology is based on the systolic design methodology [28, 27]. The designer environment is chosen by the algorithm designer. Usually algorithms are simulated in imperative languages (like C of Fortran). The translation to the Alpha format corresponds to an imperative to functional code translation [16]. In the MMAlpha environment, bit-level simulation can be carried out on the functional description of the algorithm. Then refinement of the specification is performed in the MMAlpha environment and the translation to VHDL (hardware description language) is done automatically. All this is described in section 3.2. The last part of the design is the logic synthesis which is usually performed with commercial tools (section 3.3).

3.2 Design path

The generic mathematical method, based on the robust incremental optical flow estimation for solving discontinuity energy-based applications, has been presented in equation (6,7). In this section, we present here the complete design path from this mathematical specification to the description of the architecture in VHDL.

Algorithmic level transformations

The first step of the design methodology is to translate the initial specifications (eqn (6,7)) in a functional formalism (in our case, recurrence equations [28]). We consider that the original image, f, is indexed with two indices i, j, and that $\boldsymbol{w}(i, j, n)$ is the restored image after n steps of the iteration. the algorithm is expressed according to a particular neighborhood system (say, 4-neighborhood system: North, South, East, West). If we just want to express the Jacobi iteration (without any updates of the weights or incremental registration during the iteration), this can be done with the following recurrence equations (for sake of being concise, we have not represented the initializations of variables):

$$\forall\ (i,j,n) \in \{i,j,n | 1 \le i,j \le m; 1 \le n \le P\};\quad \kappa \in \{\underbrace{(-1,0)}_{N}, \underbrace{(0,-1)}_{W}, \underbrace{(1,0)}_{S}, \underbrace{(0,1)}_{E}\}$$

$$\begin{cases} \boldsymbol{w}(i,j,n) = & \bar{\boldsymbol{w}}(i,j,n-1) - P(i,j,n-1)\nabla f(i,j) \\ P(i,j,n) = & \dfrac{\nabla f(i,j) \cdot \bar{\boldsymbol{w}}(i,j,n) + f_t(i,j)}{\alpha + \|\nabla f(i,j)\|^2} \\ \bar{\boldsymbol{w}}(i,j,n) = & \displaystyle\sum_{\kappa} \boldsymbol{w}[(i,j)+\kappa, n]/4 \end{cases} \tag{8}$$

For sake of being concise, we have only represented the Alpha program which computes the simple Jacobi iterations without updating of the weights and without incremental registration (figure 6). The others algorithms can be expressed as well but they are a little bit more complicated to read.

Introducing the estimators of the iteration consists in introducing the weights β_κ and δ_κ and modifying the definition of P and $\bar{\boldsymbol{w}}$ (Cauchy's estimator has been chosen, $\rho'(x) = \sigma^2/(\sigma^2 + x^2)$). For clarity sake, we have chosen to represent only the discontinuity weights (βs) on this recurrence equation.

$$\forall\ (i,j,n) \in \{i,j,n | 1 \le i,j \le m; 1 \le n \le P\};\quad \kappa \in \{(-1,0), (0,-1), (1,0), (0,1)\}$$

$$\begin{cases} \boldsymbol{w}(i,j,n) = & \bar{\boldsymbol{w}}(i,j,n-1) - P(i,j,n-1)\nabla f(i,j) \\ P(i,j,n) = & \dfrac{\nabla f(i,j) \cdot \bar{\boldsymbol{w}}(i,j,n) + f_t(i,j)}{\alpha \sum_\kappa \beta_\kappa(i,j,n) + \|\nabla f(i,j)\|^2} \\ \bar{\boldsymbol{w}}(i,j,n) = & \dfrac{\sum_\kappa \beta_\kappa(i,j,n)\boldsymbol{w}[(i,j)+\kappa,n]}{\sum_\kappa \beta_\kappa(i,j,n)} \\ \beta_\kappa(i,j,n) = & \dfrac{\sigma^2 \rho'(\|\boldsymbol{w}(i,j,n) - \boldsymbol{w}[(i,j)+\kappa,n]\|)}{2(\|\boldsymbol{w}(i,j,n) - \boldsymbol{w}[(i,j)+\kappa,n]\|)} \end{cases} \tag{9}$$

However, if we consider the incremental registration at each iteration, the $\boldsymbol{w}$ quantity is split into $\boldsymbol{w} + \boldsymbol{dw}$. The term $\boldsymbol{dw}(i,j,n)$ is the motion computed at step n. Each time a component of $\boldsymbol{dw}$ grows greater than one, it is decreased by one and the corresponding $\boldsymbol{w}$ is increased accordingly. The recurrence equation specification is written in (10). The equation defining $\bar{f}$ is not strictly a recurrence equation (dynamic indexing) but can transformed in by introducing nested `if` (because $\boldsymbol{inc}$ can only

```
system mouv :  {N,M,P | 3<=N; 3<=M; 3<=P}
               (fx : {i,j | 1<=i<=N; 1<=j<=M} of real;
                fy : {i,j | 1<=i<=N; 1<=j<=M} of real;
                ft : {i,j | 1<=i<=N; 1<=j<=M} of real;
                coef :  real)
       returns (u : {i,j | 1<=i<=N; 1<=j<=M} of real;
                v : {i,j | 1<=i<=N; 1<=j<=M} of real);
var
  ut,vt: {i,j,n | 0<=i<=N+1; 0<=j<=M+1; 1<=n<=P} of real;
  ub,vb,PP : {i,j,n | 1<=i<=N; 1<=j<=M; 1<=n<=P} of real;
let
  ub[i,j,n] = (ut[i-1,j,n] + ut[i+1,j,n] + ut[i,j-1,n] + ut[i,j+1,n]) / 4[];
  vb[i,j,n] = (vt[i-1,j,n] + vt[i+1,j,n] + vt[i,j-1,n] + vt[i,j+1,n]) / 4[];
  PP[i,j,n] = (fx[i,j] * ub[i,j,n] + fy[i,j] * vb[i,j,n] + ft[i,j]) /
              (coef[] + fx[i,j] * fx[i,j] + fy[i,j] * fy[i,j]);
  ut[i,j,n] =
      case
        {| n=1} | {| i=0} | {| i=N+1} | {| j=0} | {| j=M+1} : 0[];
        {| 1<=i<=N; 1<=j<=M; 2<=n} : ub[i,j,n-1] - fx[i,j] * PP[i,j,n-1];
      esac;
  vt[i,j,n] =
      case
        {| n=1} | {| i=0} | {| i=N+1} | {| j=0} | {| j=M+1} : 0[];
        {| 1<=i<=N; 1<=j<=M; 2<=n} : vb[i,j,n-1] - fy[i,j] * PP[i,j,n-1];
      esac;
  u[i,j] = ut[i,j,P];
  v[i,j] = vt[i,j,P];
tel;
```

Figure 6: *Alpha code for the motion estimation computed by the Jacobi method (without robust estimators and without incremental registration). This program corresponds to* `P` *iterations of the method on an* `N`×`M` *image.*

take the values 1,0 or -1). Note also that image f is now indexed by n, because we have to compute the derivatives $\nabla f(i,j,n)$ and $f_t(i,j,n)$ at each iterations.

$$\forall\ (i,j,n) \in \{i,j,n | 1 \le i,j \le m; 1 \le n \le P\}; \quad \kappa \in \{(-1,0),(0,-1),(1,0),(0,1)\}$$

$$\begin{cases} \boldsymbol{inc}(i,j,n) = & [if\ |\boldsymbol{dw}_l(i,j,n)| > 1\ then\ sign(\boldsymbol{dw}_l(i,j,n))\ else\ \boldsymbol{dw}_l(i,j,n)]_{l=1,2} \\ \bar{f}(i,j,n) = & f[(i,j) + \boldsymbol{inc}(i,j,n), n] \\ \boldsymbol{dw}(i,j,n) = & \boldsymbol{dw}(i,j,n-1) - \boldsymbol{inc}(i,j,n-1) \\ \boldsymbol{w}(i,j,n) = & \bar{\boldsymbol{w}}(i,j,n-1) - P(i,j,n-1)\nabla\bar{f}(i,j,n-1) + inc(i,j,n-1) \\ P(i,j,n) = & \dfrac{\nabla\bar{f}(i,j,n)\cdot\bar{\boldsymbol{w}}(i,j,n) + \bar{f}_t(i,j,n)}{\alpha\sum_\kappa \beta_\kappa(i,j,n) + \|\nabla\bar{f}(i,j,n)\|^2} \\ \bar{\boldsymbol{w}}(i,j,n) = & \dfrac{\sum_\kappa \beta_\kappa(i,j,n)(\boldsymbol{w}[(i,j)+\kappa,n] + \boldsymbol{dw}[(i,j)+\kappa,n]}{\sum_\kappa \beta_\kappa(i,j,n)} \\ \beta_\kappa(i,j,n) = & \dfrac{\sigma^2\rho'(\|\boldsymbol{w}(i,j,n) + \boldsymbol{dw}(i,j,n) - (\boldsymbol{w}[(i,j)+\kappa,n] + \boldsymbol{dw}[(i,j)+\kappa,n])\|)}{2(\|\boldsymbol{w}(i,j,n) + \boldsymbol{dw}(i,j,n) - (\boldsymbol{w}[(i,j)+\kappa,n] + \boldsymbol{dw}[(i,j)+\kappa,n])\|)} \end{cases} \tag{10}$$

We have here a recurrence equation specification which provides multi-resolution motion estimation. This was previously impossible because of the dynamic shifting of the image after each resolution. By decomposing these shifts into small shifts – i.e at most by one pixel –, we propose a solution which can be implemented in hardware.

This specification has been simulated to determine of the bit width necessary for the different variables. Most of the variables represent real numbers. As floating point operation are very expensive to implement in hardware, we tried to fix the number of bits necessary in a fixed point representation of the real number. We experimented on a single pair of images, this of course should be executed for a set of images which is representative of the images encountered.

A first rapid analysis shows that, after having normalized the data (image value) to be between 0 and 1, 4 bits of integer part where necessary to avoid overflow and the precision of the result was acceptable with 10 bits fractional part (increasing the fractional part would result in small quality improvement). As in the previous section, We measured the difference between the resulting fields by measuring the average error and standard deviation of the angular discrepancies [3] between the solution in floating point operators, and 20,14 and 11 bits fixed point operators (see table 2). A more precise study should be done to reduce the quality loss between floating point and fixed points, but the results (i.e. average number of bits by signal) will not change dramatically.

	Floating points	fixed point 20 bits	fixed point 14 bits	fixed point 11 bits
floating point	$0^o/0^o$	$2.3^o/4.9^o$	$2.7^o/4.7^o$	$5.4^o/5.2.7^o$
fixed point 20 bits	$2.3^o/4.9^o$	$0^o/0^o$	$0.6^o/0.2^o$	$3.6^o/2.4^o$

Table 2: *Comparison of motion fields obtained with different representations of the real number the motion estimation with* FAJM$_{inc}$. *the two number represent the average error and the standard deviation between the angular discrepancies [3]*

Derivation of hardware

The initial step of the design methodology (referred in the literature as *uniformization* or *localization* [11]) transforms all broadcasts into a pipeline of data values in order to obtain a locally connected architecture. Here for instance, we will need to pipeline the image variables, `fx[i,j]`, `fy[i,j]` and `ft[i,j]` to all computation points `(i,j,n)`. This process replaces (in the program of figure 6) `fx[i,j]` by `Pipefx[i,j,n]` everywhere it is used and adds the following pipeline equation:

```
Pipefx[i,j,n] =  case
       {| n=1; } : fx[i,j];
       {| 2<=n<=P; } : Pipefx[i,j,n-1];
     esac;
```

The resulting program is now uniform (every dependence vector is constant). There are no more data broadcasts. The resulting architecture involves only local communications between neighboring processing elements. The next steps, called *scheduling* and *allocation*, constitute the parallelization stages. They intrinsically govern what kind of architectures we are finally going to deal with.

Scheduling of uniform recurrence equations has been widely studied (see [9] for example); it gives an execution date for all computations. An automatic schedule procedure is provided in MMAlpha; a global clock is assumed and the execution date is given by a clock counter. The designer can specify which signal he wants to store (in a register) in one iteration of the computation (i.e. computation of `ut(i,j,n)` from `ut(i,j,n-1)`). The simplest choice (which leads to the longest clock cycle) is to set only one register on the path between `ut(i,j,n-1)` (resp. `vt(i,j,n-1)`) and `ut(i,j,n)` (resp. `vt(i,j,n)`). This yields the following schedule: every local variable of the program given in Figure 6 is computed at clock tick `n`, and `ut[i,j,n]` is stored in a register until it is used (at step `n+1`); final results `u[i,j]` and `v[i,j]`, being computed at step `P`.

The designer must now specify the allocation function. This function is usually a projection of the dependence graph. Here, the trivial projection is $(i,j,n) \to (i,j)$ which means that each virtual processor is in charge of one pixel of the restored image. In Alpha, once the schedule and projection are chosen, one can express the computation in a basis such that the first index represents time and the other indices represent processor space. This is very convenient for a structural interpretation of the code. The new indices of the resulting program are now denoted: `t,p1,p2`.

At this point, the designer can check whether or not the image is present in the array at the beginning (and whether or not the result should be output from the array). Here, the equation defining `Pipefx` is:

```
Pipefx[t,p1,p2] =  case
       {| t=1; } : fx[p1,p2];
       {| 2<=t<P; } : Pipefx[t-1,p1,p2];
     esac;
```

If we decide that the initial image `fx[i,j]` should arrive on the first column of processors (to respect I/O hardware constraint), then another pipelining transformation can be performed. The new equations for `Pipefx` are:

```
Pipefx[t,p1,p2] = case                PipeIOfx[t,p1,p2] =  case
        {| t=1} : PipeIOfx[t,p1,p2];          {| p1=0;} : fx[-t+1,p2];
        {| 2<=t<=P} : Pipefx[t-1,p1,p2];      {| 1<=p1<=N } : PipeIOfx[t-1,p1-1,p2];
      esac;                                 esac;
```

The array shown on the left of Figure 7 can be drawn at this design step. From that point onward, we have a representation of the array (size, shape) and a functional behavior of each processor of the

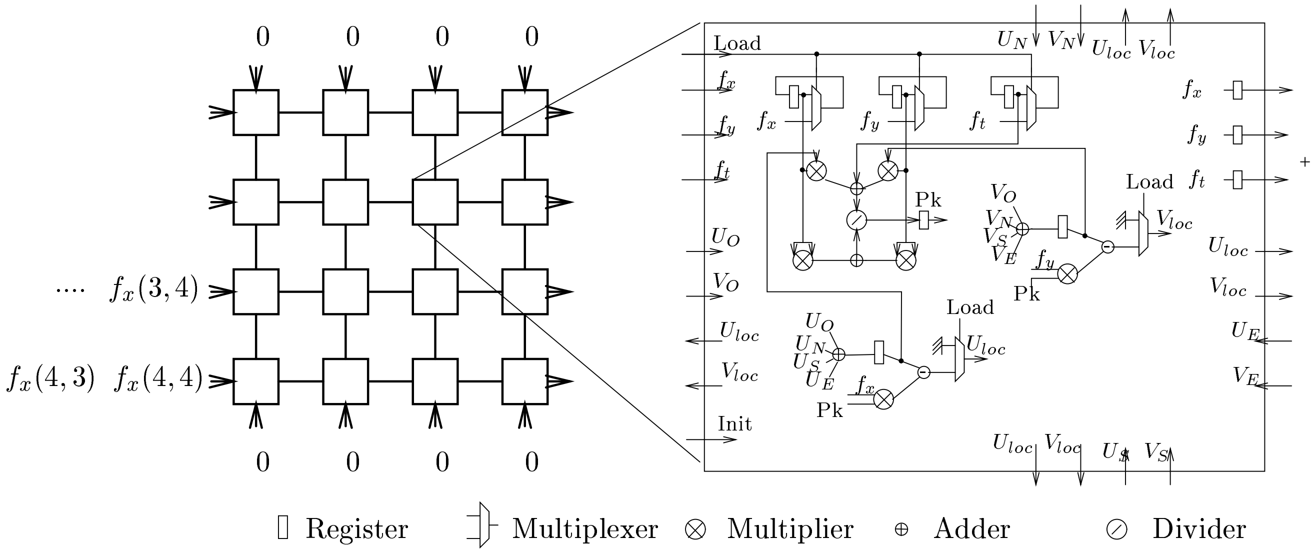

Figure 7: *Processor array (Jacobi method) for image restoration. Original image derivatives (f_x, f_y, f_t) are input on the left of the array and the result is output on the right of the array. The schematic of one cell is detailed on the right. Load is a control control signal, other signals are data. In this implementation, input and output are pipelined but the control signals (load) are broadcasted.*

array (i.e., instruction executed at each time step of the global clock). In the example of Figure 7, data enter the array from the left and results are flushed out on the right. Let us note that for the sake of clarity and simplicity we only show here the architecture obtained for the Jacobi case (without weight computation nor incremental registration).

The main difficulty in the low level transformations is the control generation. Indeed, all the controls realized with operators of high level languages must be realized in hardware. Fortunately, due to the computational regularity of uniform programs in Alpha, the control signal generation can be done automatically. In the current example of Figure 7, the *load* control signal is broadcasted (all the processors start the computation simultaneously) but the loading of the image is pipelined from left to right (signal f_x, f_y, f_t).

After the control generation step, an Alpha program which can be directly interpreted as an architecture is obtained. One possible realization here is illustrated Figure 7. All these steps have been executed by MMAlpha functions controlled by the designer with a simple command line.

VHDL generation and partitioning

As partitioning, in the current state of MMAlpha, has to be done by hand, we have not studied it in detail. However, we can evaluate a partitioning strategy. One simple way of reducing the number of processors is to split the computation of the 4×4 array of Figure 7 into, for instance, 8 pass on a 2×1 array. Or more generally, to split the computation of the $N \times N$ array into, for instance, N^2/P passes on a $P \times 1$ array. From the host computer point of view (the computer which sends the image to the hardware), the partitioned array requires a little more complicated procedure for handling Input/Output but the ratio communication/computation is roughly the same.

The parameter P is chosen such that the Input/Output data throughput respects the constraints of

design characteristics	area of 1 cell (mm^2)	Clock cycle ($10^{-9}s$)	time for convergence (s)
Jacobi (8 bits)	0.4 mm^2	37 ns	9.2 10^{-6}s
Jacobi (14 bits)	1.4 mm^2	56 ns	14 10^{-6}s
FAJM (8 bits)	9.52 mm^2	89 ns	22 10^{-6}
FAJM (14 bits)	24.79 mm^2	117 ns	29 10^{-6}s

Table 3: *Experimental results of hardware estimation for one cell of optical flow estimation architecture. The robust estimator used here is $\rho(x) = \log(1 + \frac{x^2}{\sigma^2})$. Convergence time is evaluated with 250 iterations for the Jacobi relaxation, the technology used is $\lambda = 0.6\mu m$*

the interface media between the host and the chips and that the overall speed is fast enough for real time constraints. Hence, the choice of the partitioning strategy depends on the results of the synthesis (i.e. on its implication for speed and area of the resulting architecture) and on the some other constraints like the Input/Output throughput available between the host processor and the chip. These issue will be analyzed in section 3.3.

The translation to synthesizable VHDL from this specification is automatic with MMAlpha and the synthesis can continue with commercial tools. As mentioned earlier, this design process can be easily modified in order to obtain other designs (for instance, pipelining control signals instead of broadcasting them).

3.3 Design process results

Architectures were synthesized for the motion estimation problem. The Jacobi and the FAJM methods (i.e. iterative updating of the weights) were implemented. The Cauchy M-estimator was chosen as the robust cost function.

Logic synthesis was done (on the architecture roughly depicted in Figure 7) for the Jacobi and FAJM method, with 8-bit and 14-bit (i.e. 10+4) wide signals. Table 3 shows the estimated clock cycle of the architecture and the surface area for one cell. These estimations were carried out using the Compass logic design tool on the VHDL code obtained with the MMAlpha translator. No divider was provided in the Compass library and therefore it was replaced by multipliers during the evaluations. All operators were fixed-point operators. From these estimations, a simple Jacobi relaxation for a 64×64 image would take a surface of 7.5 cm $\times$ 7.5 cm (with a motion estimation each 10^{-5} second).

The architectures derived for the FAJM method with 14 bits wide signals were too expensive in terms of surface area for realistic image sizes (i.e at least 256×256). A partitioning of the array should be considered for effectively implementing such kind of architectures.

If we want to partition our array like mentioned earlier, with $P = 16$ for instance (16 processors instead of 65536), it means that we must send 16 triplets of images every cycles. Considering that the derivatives are computed on 14 bits, this represent 84 bytes per tops, that is (with the result of table 3) 840 Mb/s. This is a little bit too much for current PCI busses norm but should be available soon (PCI 66-64 and AGP norm already provide 400 Mb/s). On the other hand, there will be $N^2/16$ use of the array for one iteration. Considering that convergence occurs after 100 steps, we will need $100N^2/16$ tops per image i.e. $N^2/16 \times 10^{-5}$ seconds per image. With $N = 256$, we obtain 0.04 seconds per image, i.e. roughly what is needed for real time processing (24 image per second). This evaluation shows that it may also be possible to use an FPGA for such treatment. Provided that the I/O throughput is obtained, the next generation of Xilinx FPGAs will contains 1 million gates which is much more than what we need for 16 cells (from the result of table 3 approximately 16 thousand gates will be needed).

It appeared that architectures integrating the incremental registration which can be synthesized with our methodology are much more complex, mainly because of the integration of the derivatives computations on chip. We would need a new high level transformation for sharing resources (mostly multipliers and adders) before providing practical results. However, we believe that multi-resolution is the main advantage of this method. Hence it worth investing a little bit more to take advantage of the results of section 2.1.

4 Conclusion

We proposed a design methodology for hardware synthesis of motion estimation. Following the work presented in [25], where we introduced the FAJM method, we proposed here the FAJM$_{inc}$ method which permits executing multi-resolution motion estimation in hardware. The new non-linear method (FAJM$_{inc}$) has been intensively studied from the convergence speed and quality of result point of view on a benchmark of images. The experimental comparison between this method and standard minimization techniques in that context demonstrates its efficiency and relevance.

Bit level simulation was implemented on these algorithms which indicated the approximate bit width necessary for a hardware implementation. Synthesis of hardware was implemented through a high level VLSI derivation tool for a subset of the algorithms described (those not containing the incremental registration). Practical evaluation of the performances of the obtained circuits (in terms of area and clock cycle) where presented. These result show that FPGA implementation of these algorithm will be useful in the next generation of machine (Xilinx's Virtex for instance). This is an important issue as FPGA co-processors will probably soon be available on general purpose machines.

The complexity of the architecture generated is inherent to the class of algorithms chosen in this paper. However, recent research in computer vision tends to focus on this kind of algorithm [30], justifying the choice of these hardware implementations. Finally, the derivation technique that we used for VLSI design can also be used for FPGA design. Indeed the regularity of the design and the automation of the process are very useful if we target a fast implementation on FPGA.

The technological gap with which we are confronted led to the idea of integrating a processor *behind* a sensor [29], and the concept of *smart sensors* [14] and *Active Pixel Sensor* [26] led to numerous achievements. Bi-dimensional arrays of such sensors have already been implemented [2]. However, because of current technology restrictions, these smart sensors are often implemented with full custom design (often in analog technology), which is time consuming and error prone. We believe that the high level design methodologies that have been developed for digital technology enable the achievement of more secure designs. Moreover, in a few years, when the technology allows greater integration, the power consumption and price will be advantageous for digital technology.

References

[1] P. Anandan. – A computational framework and an algorithm for the measurement of visual motion.

– *Int. Journ. of Comp. Vision*, 2:283–310, 1989.

[2] A.G. Andreou and K.A. Boahen. – A 590,000 transistors 48,000 pixel, contrast sensitive edge enhancing, cmos imager-silicon retina. – In *Proc. 16th Conf. Advanced Research in VLSI*, pages pp. 225–240, 1995.

[3] J. Barron, D. Fleet, and S. Beauchemin. – Performance of optical flow techniques. – *Int. J. Computer Vision*, 12(1):43–77, 1994.

[4] J. Bergen, P. Anandan, K. Hanna, and R. Hingorani. – Hierarchical model-based motion estimation. – In G. Sandini, editor, *Proc. Europ. Conf. Computer Vision*, volume 558 of *LNCS*, pages 237–252. Springer-Verlag, 1992.

[5] M. Black and P. Anandan. – The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. – *Computer Vision and Image Understanding*, 63(1):75–104, 1996.

[6] M. Black and A. Rangarajan. – On the unification of line processes, outlier rejection, and robust statistics with applications in early vision. – *Int. J. Computer Vision*, 19(1):75–104, 1996.

[7] A. Blake and A. Zisserman. – *Visual Reconstruction.* – The MIT Press, Cambridge, USA, 1987.

[8] P. Charbonnier, L. Blanc-Féraud, G. Aubert, and M. Barlaud. – Deterministic edge-preserving regularization in computed imaging. – *IEEE Trans. Image Processing*, 6(2):298–311, 1997.

[9] Alain Darte and Yves Robert. – Constructive methods for scheduling uniform loop nests. – *IEEE Transactions on Parallel Distributed Systems*, 5.8:814–822, 1994.

[10] A. Delanay and Y. Bresler. – Globally convergent edge-preserving regularized reconstruction: an application to limited-angle tomography. – *IEEE Trans. Image Processing*, 7(2):204–221, February 1998.

[11] Vincent Van Dongen and Patrice Quinton. – Uniformization of linear recurrence equations: a step towards the automatic synthesis of systolic array. – In K. Bromley et al. eds., editor, *International Conference on Systolic Arrays*, pages 473–482. IEEE Computer Society Press, 1988.

[12] W. Enkelmann. – Investigation of multigrid algorithms for the estimation of optical flow fields in image sequences. – *Comp. Vision Graph. and Image Proces.*, 43:150–177, 1988.

[13] D. Fleet and A. Jepson. – Computation of component image velocity from local phase information. – *Int. Journ. of Comp. Vision*, 5(77-104), 1990.

[14] R. Forchhmeimer and A. Astrom. – Mapp2200: a second generation smart optical sensor. – In *Proc. SPIE, Vol. 1659, Image Processing and Interchange*, 1992.

[15] D. Geman and G. Reynolds. – Constrained restoration and the recovery of discontinuities. – *IEEE Trans. Pattern Anal. Machine Intell.*, 14(3):367–383, 1992.

[16] P.C. Held. – Hipars: a tool for automatic conversion of nested loop programs into single assignment programs. – Technical Report Dept. Electrical Engineering, Delft University of Technology, 1994.

[17] P. Holland and R. Welsch. – Robust regression using iteratively reweighted least-squares. – *Commun. Statis.-Theor. Meth.*, A6(9):813–827, 1977.

[18] B. Horn and B. Schunck. – Determining optical flow. – *Artificial Intelligence*, 17:185–203, 1981.

[19] P. Huber. – *Robust Statistics.* – John Wiley & Sons, 1981.

[20] B. Jahne, H. Hauβecker, H. Spies, D. Schmundt, and U. Schurr. – Study of dynamical processes with tensor-based spatiotemporal image processing techniques. – In *Proc. Europ. Conf. Computer Vision*, Freiburg, Germany, 1998.

[21] S. Lai and B. Vemuri. – Reliable and efficient computation of optical flow. – *Int. Journ of Comp. Vision*, 29(2):87–105, 1998.

[22] C. Mauras. – *Alpha : un langage équationnel pour la conception et la programmation d'architectures parallèles synchrones.* – PhD thesis, Université de Rennes 1, IFSIC, December 1989.

[23] E. Mémin and P. Pérez. – Dense estimation and object-based segmentation of the optical flow with robust techniques. – *IEEE Trans. Image Processing*, 7(5):703–719, 1998.

[24] E. Mémin and T. Risset. – Hardware driven considerations for energy based applications. – Technical Report 1220 (accepté sous condition de révisions mineures à Real Time Imaging), IRISA, April 1999.

[25] E. Mémin and T. Risset. – Vlsi design methodology for edge-preserving image reconstruction. – *Real-Time Imaging*, accepted for publication.

[26] S. Mendis, S.E. Kemeny, and E.R. Fossum. – Cmos active pixel sensor. – *IEEE Trans. on Electon. Devices*, 41(3):452–453, march 1994.

[27] D.I. Moldovan. – On the analysis and synthesis of vlsi systolic arrays. – *IEEE Transactions on Computers*, 31:1121–1126, 1982.

[28] Patrice Quinton and Yves Robert. – *Systolic Algorithms and Architectures.* – Prentice Hall and Masson, 1989.

[29] J. Tanner and C. Mead. – An integrated analog optical motion sensor. – In S.Y. Kung, R. Owen, and J.G. Nash, editors, *VLSI signal processing II.* IEEE Press, 1986.

[30] B. M. ter Haar Romeny, editor. – *Geometry-driven diffusion in computer vision.* – Kluwer academic, 1994.

[31] D. Wilde. – The alpha language. – Technical Report 827, IRISA, Rennes, France, Dec 1994.