High performance motion detection: some trends toward new embedded architectures for vision systems

Lionel Lacassagne, Antoine Manzanera, Julien Denoulet, Alain Mérigot

Journal of RealTime Image Processing manuscript No.
(will be inserted by the editor)

Lionel Lacassagne · Antoine Manzanera · Julien Denoulet · Alain Mérigot

High Performance Motion Detection:

Some trends toward new embedded architectures for vision systems

Received: date / Revised: date

Abstract The goal of this article is to compare some optimized implementations on current High Performance Platforms in order to highlight architectural trends in the field of embedded architectures and to get an estimation of what should be the components of a *next generation* vision system. We present some implementations of robust motion detection algorithms on three architectures: a general purpose RISC processor - the PowerPC G4 - a parallel artificial retina dedicated to low level image processing - *Pvlsar34* - and the Associative Mesh, a specialized architecture based on associative net. To handle the different aspects and constraints of embedded systems, execution time and power consumption of these architectures are compared.

Keywords

Embedded system, Vision-SoC, RISC, SIMD, SWAR, parallel architecture, programmable artificial retina, associative nets model, vision, image processing, motion detection, High Performance Computing.

Lionel Lacassagne
Institut d'Electronique Fondamentale (IEF/AXIS)
Université Paris Sud
E-mail: lionel.lacassagne@u-psud.fr

Antoine Manzanera
Laboratoire d'Electronique et d'Informatique
Ecole Nationale Supérieure de Techniques avancées (ENSTA)
E-mail: antoine.manzanera@ensta.fr

Julien Denoulet
Laboratoire des Instruments et Systèmes d'Ile de France (LISIF/SYEL)
Université Pierre et Marie Curie
E-mail: julien.denoulet@upmc.fr

Alain Mérigot
Institut d'Electronique Fondamentale (IEF/AXIS)
Université Paris Sud
E-mail: alain.merigot@u-psud.fr

Introduction

For more than thirty years, Moore's law had ruled the performance and the development of computers: speed and clock frequency were the races to win. This trend slowly drifted as the processing power of computers reached a seemingly stable value. Other constraints (static current consumption, leakage, less MIPS per gates and less MIPS per Watts) of current technology - 90 nm and 65 nm - gave researchers an impulse to look for innovative directions to improve efficiency and performance of their architectures. Current challenge is to tackle power consumption to increase systems autonomy. Such technology, like IP core within embedded systems, make the processor frequency adaptable and lead to a finely optimized energy consumption. As image processing and computer vision are very CPU demanding, we focus on the impact of the architecture for a frequently used class of algorithms: the motion detection algorithms.

Three architectural paradigms are compared:

- *SWAR*: SIMD Within A Register: the impact of the SIMD multimedia extension inside RISC processors, to enhance performance;
- programmable artificial retina: one elementary processor per pixel for cellular massively parallel computation and low power consumption;
- associative net: impact of reconfigurable graph/net between processors for local and global computations and also the impact of asynchronous processors on power consumption.

We focus on the advantages and limitations of these architectures through a set of benchmarks. We also show how to modify the algorithms to take advantage each architecture's specificities. We provide different performance indexes like speed, energy required and a *"down-clocking"* frequency to enforce real-time execution. Such indexes provide insight on future trends in computer architecture for embedded systems.

The paper is organized as follow. The first section introduces a set of motion detection algorithms: Frame Difference, Markovian relaxation, Sigma-Delta algorithm and post-processing morphological operators. The second section presents the three architectures: the PowerPC G4, *Pvlsar34* (a 200×200 Programmable Artificial Retina) and the Associative Mesh (an asynchronous net with SIMD functional units). This section also provides details about how algorithms are optimized in regard to the targeted architectures. The third section deals with benchmarking: benchmark of the different algorithms in term of speed and in term of power consumption. To conclude, a synthesis of two extensive benchmarks is provided.

1 Motion detection algorithms

As the number of places observed by cameras is constantly increasing, a natural trend is to eliminate the human interaction within the video monitoring systems and to design fully automatic video surveillance devices. Although the relative importance of the low level image processing may vary from one system to the other, the computational weight of the low level operators is generally high, because they involve a great amount of data. Thus, the ability of video surveillance systems to detect a relevant event (intrusion, riot, distress...) is strongly related to the performance of some crucial image processing functions.

Such fundamental processing step is the motion detection, whose purpose is to partition the pixels of every frame of the image sequence into two classes: the *background*, corresponding to pixels belonging to the static scene (label: 0) and the *foreground*, corresponding to pixels belonging to a moving object (label: 1). A motion detection algorithm must discriminate the moving objects from the background as accurately as possible, without being too sensitive to the sizes and velocities of the objects, or to the changing conditions of the static scene. For long autonomy and discretion purposes, the system must not consume too much computational resources (energy and circuit area). The motion detection is usually the most computationally demanding function of a video surveillance system. How the algorithm is actually computed and on which architecture, then become crucial questions.

Three algorithm/architecture pairs will be considered here. In order to compare those very different architectures, we will consider different versions of motion detection algorithms with similar quality but relying on different computational models, some of them being more adapted to one architecture than the other.

The motion detection algorithm can be separated into two parts: (1) time-differentiation and (2) spatiotemporal regularization.

The purpose of the time-differentiation part is to provide, for every pixel x and every time index t: a measure of the temporal variation (the observation), denoted O_t and an initial value of the motion binary label, denoted $\hat{E}_t$. The "frame difference" option is classical and fairly obvious: the temporal derivative is approximated by a difference between consecutive frames, whose absolute value is used as a single motion map (observation) $O_t(x) = |I_t(x) - I_{t-1}(x)|$ and the initial value of the motion label $\hat{E}_t$ is obtained by thresholding O_t. The "SigmaDelta" option – detailed in Section 1.1 – is a recent algorithm (29), based on non-linear estimation of temporal statistics of every pixel.

The spatiotemporal regularization part aims at exploiting the correlations between neighboring pixels in the motion measures in order to improve the localization of the moving objects. Two main options are considered here: (a) Morphological filtering, detailed in Section 1.2 and (b) Markovian relaxation, detailed in Section 1.3.

So, the "Sigma-Delta" can be seen as a pre-processing step for the Markovian regularization or as the main algorithm when followed by a morphological post-processing.

1.1 Sigma-Delta Estimation

The principle of the $\Sigma\Delta$ algorithm is to estimate two parameters M_t and V_t of the temporal signal I_t within every pixel using $\Sigma\Delta$ modulations. It is composed of four steps: (1) update the current background image M_t with a $\Sigma\Delta$ filter, (2) compute the frame difference between M_t and I_t, (3) update the time-variance image V_t from the difference O_t using a $\Sigma\Delta$ filter and (4) estimate the initial motion label $\hat{E}_t$ by comparing the current difference O_t and time-variance V_t.

for each pixel x:
if $M_t(x) < I_t(x), M_t(x) = M_{t-1}(x) + 1$
if $M_t(x) > I_t(x), M_t(x) = M_{t-1}(x) - 1$
otherwise $M_t(x) = M_{t-1}(x)$

step1: update M_t

for each pixel x:
$O_t(x) = |M_t(x) - I_t(x)|$

step2: compute O_t

for each pixel x such that $O_t(x) \neq 0$:
if $V_t(x) < N \times O_t(x), V_t(x) = V_{t-1}(x) + 1$
if $V_t(x) > N \times O_t(x), V_t(x) = V_{t-1}(x) - 1$
otherwise $V_t(x) = V_{t-1}(x)$

step3: update V_t

for each pixel x:
if $O_t(x) < V_t(x)$
then $\hat{E}_t = 0$
else $\hat{E}_t = 1$

step4: estimate $\hat{E}_t$

Apparently, the only parameter is the amplification factor N of the difference (typical values of N are in $2 \ldots 4$). The dimension of N is the number of standard deviation used in the initialization of the motion label. In fact, the updating frequency, which has the dimension of number of gray level per second, can also be adapted. This is a way of customizing the $\Sigma\Delta$ estimation to different kinds of motion and image noise (30).

$\varepsilon_B(I)(x) = \bigwedge_{b\in B} I(x-b)$ $\delta_B(I)(x) = \bigvee_{b\in B} I(x+b)$ (a)	$\gamma_B = \delta_B \circ \varepsilon_B$ $\varphi_B = \varepsilon_B \circ \delta_B$ (b)
$\xi_B = \varphi_B \circ \gamma_B$ $\theta_B = \gamma_B \circ \varphi_B$ (c)	$\Xi_n = \xi_{B_n} \circ \Xi_{n-1}$ $\Theta_n = \theta_{B_n} \circ \Theta_{n-1}$ (d)

Table 1 Morphological operators: (a) Erosion and dilatation (b) Opening and closing (c) Alternate filters (d) Alternate Sequential filters.

1.2 Morphological filtering

1.2.1 Alternate Sequential Filters (ASF)

The first option of morphological filtering is to perform a sequence of dilatations and erosions using a set of structuring elements of increasing size, such as a sequence of discrete balls $(B_n)_n$, $B_n = \{z \in \mathbb{Z}^2; d(z, O) \leq n\}$, with O the origin of the discrete plane $\mathbb{Z}^2$ and d a discrete distance of $\mathbb{Z}^2$. Table 1 shows the definitions of such operators.

$\wedge$ and $\vee$ respectively represent the logical `AND` and `OR`. By convention, Ξ_0 and Θ_0 both correspond to the identity function. In this option, the spatiotemporal regularization is performed by applying an alternated sequential filter of certain size to the output of the temporal detection. Typically, $E_t = \Xi_2(\hat{E}_t)$.

1.2.2 Density operators

In a similar fashion, density operators are defined using a structuring element B, except that the binary response is based on counting-thresholding instead of `AND-OR` combinations :

$$\mathcal{D}_B(I)(x) = 1 \Longleftrightarrow |\{b; I(x-b) = 1\}| \geq \theta$$

where $|S|$ represents the cardinality of set S and θ a threshold representing a required density of 1s. In this case, the final label is computed using a density operator with a ball of radius n: $E_t = \mathcal{D}_{B_n}(\hat{E}_t)$. Typically n equals 1, 2 or 3 and usually $\theta = \lceil |B_n|/2 \rceil$ (majority voting).

1.2.3 Geodesic reconstruction

Defined from a binary reference image R, the geodesic reconstruction $Rec^R(I)$ of Image I within Reference R is the relaxation of the geodesic dilatation of I within R: $\delta_B^R(I) = \delta_B(I) \wedge R$. Assuming that the structuring element B - basically a discrete ball of radius 1 - is defining the topology, $Rec^R(I)$ corresponds to the connected components of R having a non-empty intersection with I.

In this option, the final label E_t is computed as follows: small connected components elimination using an opening by reconstruction with a ball of radius n: $\tilde{E}_t = Rec^{\hat{E}_t}(\gamma_{B_n}(\hat{E}_t))$, then temporal confirmation by computing another reconstruction: $E_t = Rec^{\tilde{E}_t}(\tilde{E}_{t-1})$.

The final motion label E_t then corresponds to the objects (connected components) bigger than B_n that appear on two consecutive frames.

1.3 Markovian Relaxation

Markov Random Field based algorithms (MRF) have asserted themselves in a lot of image processing areas for regularizing ill-posed problems. Albeit robust, their well-known drawback is their CPU consumption due to a large amount of computations, which led researchers to look for solution to speed up its execution time, using parallel machines or dedicated architectures (1; 2; 8; 19; 28).

We follow the MRF model introduced for motion detection purposes proposed by the LIS-Grenoble laboratory (6) and derived from the IRISA model (4; 26). This model is based on the estimation of a binary (background/foreground) motion field e given an *observation field* o, by maximizing a Bayesian *maximum a posteriori* criterion, i.e. given a realization of the observation field $o = y$, finding the realization x of the motion label field e that maximizes the conditional probability $P(e = x/o = y)$. Assuming that e is a Markov Random Field, linked to o with a probabilistic relation, this corresponds to finding the motion field e that minimizes the global *energy* function defined over the set of pixels $\mathbb{S}$ as follows:

$$U = \sum_{s\in\mathbb{S}} [U_m(e(s)) + U_a(e(s), o(s))],$$
$$\text{with } U_m(e(s)) = \sum_{r\in\mathcal{V}(s)} V_e(e(s), e(r)),$$
$$\text{and } U_a(e(s), o(s)) = \tfrac{1}{2\sigma^2}[o(s) - \Psi(e(s))]^2.$$

$U_m(e(s))$ is called *model energy* and is designed to provide spatiotemporal regularity in the motion field. It is based on the Markovian modeling of e as a Gibbs field, where $\mathcal{V}$ is the set of neighbors of the pixel s and the potential functions $V_e(e(s), e(r))$:

$$V(e_s, e_r) = \begin{cases} -\beta_{sr} \text{ if } e_s = e_r \\ +\beta_{sr} \text{ if } e_s \neq e_r \end{cases}$$

The β_{sr} are positive constants whose values depend on the nature of the neighborhood. We use a uniform 10-connected spatiotemporal topology (see Figure 1), with 3 different values $\beta_s = 20$ for the 8 spatial neighbors, $\beta_p = 10$ for the past neighbor and $\beta_f = 30$ for the future neighbor. Experimental tests demonstrate that these parameters do not have to be tuned according to the image sequence.

$U_a(e(s), o(s))$ is called *fitness energy* and is designed to ensure a certain level of attachment to the input data, i.e. the observation o. This term comes from the conditional probability of the observation field o, with respect to the motion field e, assuming that $o(s) = \Psi(e(s)) + n(0, \sigma^2)$, with $n(0, \sigma^2)$ a centered Gaussian noise of variance σ^2, $\Psi(e(s)) = 0$ if $e(s)$ has the background value and $\Psi(e(s)) = \alpha$ if $e(s)$ has the foreground value. The α parameter can be set to

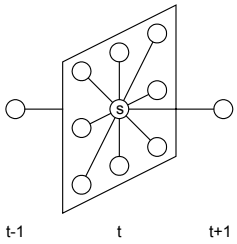

Fig. 1 spatiotemporal topology

usual value 20, or updated on the fly, as the average value of the moving observations. σ^2, the variance of the moving observation, is computed for every frame.

The minimization of the global energy U is realized by the deterministic relaxation called Iterated Conditional Mode (ICM): all the pixels are sequentially updated and each pixel s is given the label $e(s)$ corresponding to the smallest local energy $U_m(e(s)) + U_a(e(s), o(s))$. Usually, instead of a true relaxation, a limited number of scans is performed (typically 4). The advantage is that the computation time becomes independent of the data, in particular of the initial value of the motion field e.

But the drawback is that the quality of the final labeling is very dependent on that initial value, which must be close enough to the final solution. In our algorithm, we use the output of the $\Sigma\Delta$ temporal differentiation $\hat{E}_t$, which as proved a good choice of initial guess (29). The observation field o corresponds to the difference map O_t.

2 Architectures and their optimizations

In order to perform a fair comparison of these architectures, the algorithm must be optimized for each one. This section describes how the different algorithms are implemented on the three architectures, the impact of the architecture on the algorithm and how the algorithms' structure and the architectures themselves should be modified to obtain optimized implementation.

2.1 PowerPC

The powerPC used is a PPC 7447 running at 1 GHz. It has a 32 KB L1 cache, a 512 KB L2 cache and its power consumption is 10 Watt. Its specifications are detailed in tables 5 and 6. From a functional point of view (figure 2), it has one Load/Store Unit, one ALU, one FPU and a superscalar SWAR unit: Altivec. Altivec is a multimedia instruction set extension which has been designed to efficiently accelerate image and signal processing (13) applications. Altivec is composed of four 128-bit SWAR units (following the Freescale vocabulary):

- Vector Permute Unit, that handles the instructions to rearrange data within SWAR registers (permutation, selection),
- Vector Simple Integer Unit, that handles all the fast and simple integer instructions,
- Vector Complex Integer Unit, that handles the slower and complex instruction like multiply, multiply-add,
- Vector Floating Point Unit, that handles all the SWAR floating-point instructions.

Main advantages of Altivec are:

- each of the four vector units are pipelined,
- two instructions from the four units can be issued per cycle without constraint on which unit is used.

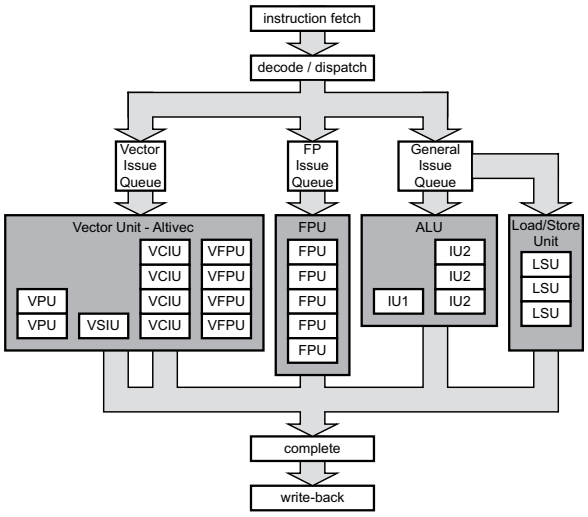

Fig. 2 PowerPC G4 pipeline

To optimize a given code for a SWAR RISC processor, we have to address the following points:

- bandwidth problem: by optimizing loads and data reuse, avoiding data reload and optimizing cache locality (34),
- pipeline stalls due to un-predictable test instructions: by trying to remove tests from MRF energy computation,
- pipeline throughput with loop transformations.

In this section we focus on SWAR optimization (also known as *SIMDization*) and algorithm transformations. Details about loop transformation techniques and above optimizations are given in (12).

To speed up the computation of the model energy u_m, we have to transform the equation of the potential function V, that comes from the Ising model (the spin associated with a particle). Usually spin-up and spin-down are coded with $+1$ / -1. In our case, rather than labeling the state of a site $-1, +1$ for *background* or *motion* pixel, we use the binary code $0, 1$. Let p_1, s_1 and f_1 the number of sites, connected to e_s, with a value 1, in the past, present and future images ($p_1 \in \{0, 1\}$, $s_1 \in \{0, \ldots, 8\}$, $f_1 \in \{0, 1\}$). Then the energy model can be computed without any test or comparison:

$$u_{m1} = (8-2s_1)\beta_s + (1-2p_1)\beta_p + (1-2f_1)\beta_f, u_{m0} = -u_{m1}$$

where u_{m1} is the energy associated to a central site at 1. The fitness energy can also be computed without taking into account the state of the site:

$$u_{a0} = \frac{1}{2\sigma^2}\left[o(s)\right]^2, \quad u_{a1} = \frac{1}{2\sigma^2}\left[o(s)-\alpha\right]^2$$

If $u_{m1} + u_{a1} < u_{m0} + u_{a0}$, the state is set to 1 otherwise it is set to 0. The change is performed whatever the previous state was. The same approach is used to remove tests from the $\Sigma\Delta$ algorithm which is actually very hard to optimize since only a few additions and comparisons are done compared to the amount of memory accesses, as described in (12). Note that this test can be optimized by rewriting $2u_{m1} < u_{a0} - u_{a1}$:

$$u_{m1} < \delta u_a, \quad \delta u_a = \frac{u_{a0}-u_{a1}}{2} = \frac{\alpha(2o-\alpha)}{4\sigma^2}$$

2.1.1 Density and opening

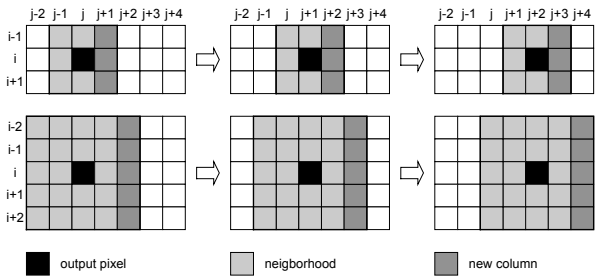

Fig. 3 operators with neighborhood: overlapping for 3×3 and 5×5 kernels

We have implemented three kernel size for these operators: 3×3, for regular use, 5×5 and 7×7 to estimate the adequacy of the considered architectures to those well known kernel operators. The cardinal computation of the ball of diameter k, that is the summation of pixel value over the $k \times k$ kernel (figure 3) requires k^2 `LOAD`, 1 `STORE` and $k^2 - 1$ mathematical operations (typically `ADD` but could be `AND` or `OR` Boolean operator for erosion and dilatation). Taking into account that $k \times k$ kernels overlap from one iteration to another one, such summation can be optimized by splitting this summation into k columns summation. The cardinal is then the sum of these k columns. For the next iteration, only one new column should be computed and added to the previous one. The new complexity is k `LOAD`, 1 `STORE` and only $2(k-1)$ `ADD` (see table 2).

For SWAR computation, the same optimizations can be applied except that 16 pixels are computed in parallel instead of only one SWAR results are given in table 2 for 16 pixels). SWAR implementation requires the construction of unaligned vector registers to compute the partial sums. This is quickly done thanks to the dedicated Altivec instruction `vec_sld` (figure 4). For example, given three aligned vector

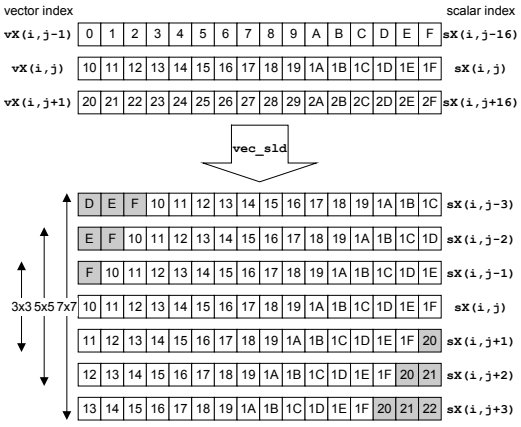

Fig. 4 SWAR convolution reuse

instruction	without split	with split
scalar `LOAD`	k^2	k
scalar `STORE`	1	1
scalar mathematical Op	k^2-1	$2(k-1)$
SWAR `LOAD`	$3k$	k
SWAR `STORE`	1	1
SWAR mathematical Op	k^2-1	$2(k-1)$

Table 2 instructions per point, scalar & SWAR version

registers (figure 4) $vX(i,j-1)$, $vX(i,j)$, $vX(i,j+1)$, the $k = 7$ unaligned vector registers sX are constructed and used to compute the sum of line i, (same computations are performed for lines $i-3, i-2, \ldots, i+3$). The interesting fact is that, up to a value of $n = 16$, pixels $x(i,j-n)$ and $x(i,j+n)$ are in the left and right vector registers of $x(i,j)$ (figure 4). So, for kernels size up to $k \times k = 33 \times 33$, only $3 \times k$ SWAR `LOAD` are required.

2.2 Programmable Artificial Retina

The purpose of the Programmable Artificial Retina (PAR) project is to develop *versatile*, *real-time*, *compact* and *low-power* vision systems. In the vision machines today, most of the resource consumption is due to the transfer of huge amounts of data throughout the different parts of the system. The data flow thus becomes the main source of cost in time, circuit area and/or energy. In PAR based vision systems, the data transfers are limited to the minimum, by processing the information where it is acquired, i.e. within every pixel of the sensor and by performing an information reduction in the focal plane in order to extract only a few descriptors representing a very small data flow, that can be processed by a low-power external processor.

The PAR concept originates from the NCP (Neighborhood Combinatorial Processing) retinas (40) which were SIMD Boolean machines. The NSIP (Near Sensor Image Processing) concept (18) then allowed to process gray level images. Now, the deep sub-micron level of CMOS technology allows

to put more and more powerful processing circuitry aside the photo receptors while preserving good acquisition performance and resolution (24) (38). The circuit used in our work was designed by T. Bernard at ENSTA and fabricated using 0.35 μm technology: *Pvlsar34* is a 200×200 retina, with an elementary digital processor and 48 bits of memory within every pixel. The architecture of *Pvlsar34* is presented in Section 2.2.1, and the retinal algorithms in Section 2.2.2 Now, whereas this architecture has proved well adapted to low and medium level image processing (33) (30), the interest of asynchronism has been identified to enhance the processing power of the PARs by providing them with a higher (i.e. regional) level of computation (16) (21) (22). This is discussed in Section 2.2.3.

2.2.1 Retina and cortex architecture

The detection algorithm presented in this paper was actually implemented on the architecture presented in figure 5. The PAR *Pvlsar34* is a CMOS sensor and a parallel machine at the same time. It is a grid of 200×200 pixels/processors connected by a regular 4-neighbors rectangular mesh. The processors execute synchronously, on their local data, a sequence of instructions sent by the controller, which is the NIOS processor IP core of the Excalibur FPGA chip. The host processor or cortex is the ARM processor hardware core of the Excalibur. It can exchange data with the PAR, modify the program sent by the NIOS to the PAR and is in charge of the higher levels of computation (i.e. non-image processing) of the vision task.

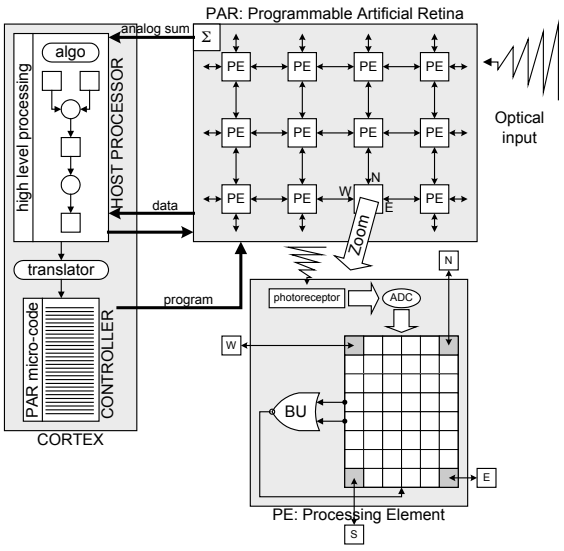

Fig. 5 Architecture of the system composed of the PAR and cortex, with focus on one elementary processor

Every pixel/processor of the PAR is composed of:

- one photo-receptor,
- one Analog to Digital Converter,
- 48 bits of digital memory,
- one Boolean Unit (BU), which can read some bits of the digital memory, compute a Boolean operation and write its output on one bit of the digital memory.

The actual instruction set of *Pvlsar34* is composed of only five instructions. If reg_1 and reg_2 are two binary registers of the digital memory:

- **one;** The BU takes the logical value `1`.
- **rd**(reg_1)**;** The BU takes the logical value of register reg_1.
- **ror**(reg_1,reg_2)**;** The BU takes the logical value of the binary `OR` between the two binary registers reg_1 and reg_2.
- **wd**(reg_1)**;** The BU writes down its logical value on register reg_1.
- **wc**(reg_1)**;** The BU writes down the complementary of its logical value on register reg_1.

Boolean algebra shows that this instruction set is sufficient to compute any Boolean function. Now, for readability purposes, we shall use in the presentation of the primitives a generic Boolean instruction set, made of the instructions of the form $y = OP(x_1, x_2)$, where x_1, x_2, y are 3 bits (not necessarily distinct) of the digital memory and OP is any binary Boolean function (e.g. `AND`, `XOR`, `ADD` `NOT`, etc). Note that every instruction is computed in a massively parallel SIMD mode, the operators are then performed simultaneously on all pixels.

Every pixel of the PAR shares 1 bit of its memory with each one of its 4 closest neighbors, allowing spatial interactions and image translations. Regarding data extraction, there are two ways to output information from the PAR:

- by translating the image and reading the output on the edge of the grid, to get the exact content of one or more bit planes of the digital memory.
- by using the Analog Global Summer, which provides in constant time an approximate measure of the number of 1s in any bit plane of the digital memory.

Although simple, this last feature is important as it provides efficiently global measures that are very useful to get spatial statistics or to detect the convergence of relaxation algorithms.

2.2.2 Cellular synchronous algorithms

From the architecture presented above, it turns out that the retinal algorithmics at the present time is essentially a cellular SIMD parallelism. A retinal program is a sequence of binary Boolean instructions. All the pixels/processors perform the same instruction at the same time on their own data, part of which can be taken from one of their closest neighbors. The extreme level of granularity and the small amount of digital memory are the main characteristics of the retinal algorithmic. The algorithm designer is imposed a constant effort of logic minimization, in order to find the Boolean expression of its algorithm that minimizes the number of elementary instructions (related to the computation time) while

fitting in the available memory (just like a hardware designer will make a circuit trying to minimize the critical paths and using the minimal amount of logical gates).

Naturally, the memory limitations also affect the data representation that can be used by the algorithm. In the case of motion processing which concerns this paper, this means that the memory used to represent the past history of every pixel must be rigorously controlled. Typically, we shall not keep histograms nor a large set of past values within every pixel, but rather a limited number of temporal statistics, computed recursively.

Despite these constraints, the retinal computation model offers some very attractive features. In particular, the fusion of acquisition and processing functions allows a close adaptation to the lighting conditions and to the scene dynamics. More precisely, the Analog to Digital Conversion (ADC) performed at the output of the photo-receptor is done by a multiple reading which provides N binary images (level sets). As the ADC itself is fully programmable, it is possible to perform a constant feedback from the local and global computations to the acquisition, thus providing sophisticated adaptation to lighting conditions.

Once the gray levels of the image are coded within every pixel of the PAR, the retinal program applies a sequence of arithmetic and logic operations that are completely written in software, at the bit level. We now present such program in the particular case of the motion detection.

The $\Sigma\Delta$ change detection algorithm relies on very simple primitives: comparison, difference and elementary increment and decrement. Furthermore, it is based on non-linear computations which does not involve neither truncation nor dynamics increasing. It is thus well adapted to the minimal instruction set and the small memory of the PAR elementary processors. The implementation on *Pvlsar34* was performed using the four primitives presented in table 3. To avoid confusing notation, I_t is noted here X_t.

Table 3(3) represents the strict comparison primitive between X_t and M_t. e and f are the two bits of result, indicating whether $M_t < X_t$ and whether $X_t < M_t$, respectively. These indicators are used in the $\Sigma\Delta$ algorithm, to update the statistics, by decrementing e (Table 3(4)) and incrementing f (Table 3(5)). Table 3(1) shows the computation of the difference O coded on n bits $\{o_0, \ldots, o_{n-1}\}$, between the current mean M_t and the current sample X_t. At the end of the computation, O_t is coded in classical 2-complement, with c the sign bit. For the second order statistics ($\Sigma\Delta$ variance V_t), it is necessary to compute the absolute value of the difference O_t (Table 3(2)).

The above primitives allow to implement the whole temporal (pixel-wise) part of the algorithm. On *Pvlsar34*, it was completed by using binary morphology as spatial regularization. An alternate sequential filter was applied on the temporal output $E_t = \Xi_2(E_t)$ (see section 1.2.1). So the only algorithmic primitives that are needed are the logical `OR` and the logical `ADD` between one pixel and its immediate neighbor, in each of the 4 directions. The filtered output E_t represents

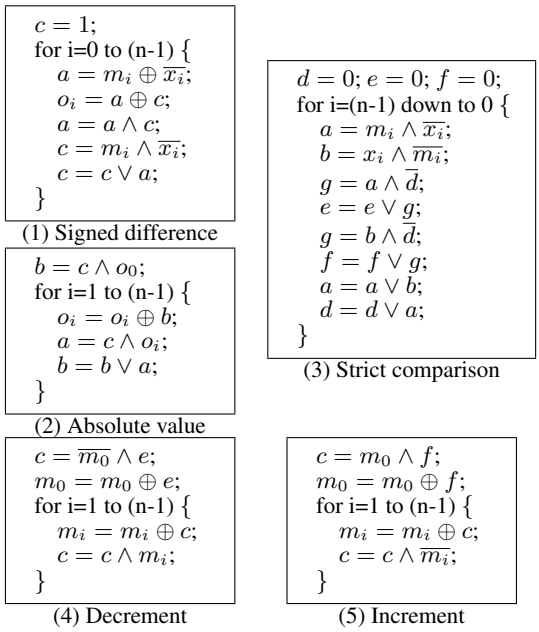

Table 3 The PAR algorithmic primitives used in the $\Sigma\Delta$ motion detection.

the final detection label and it is used as a binary mask to inhibit the update of the $\Sigma\Delta$ mean M_t.

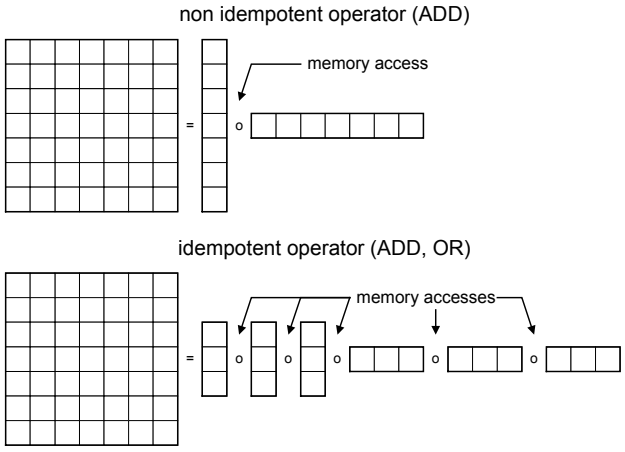

Fig. 6 2D spatial filters optimization on retina

The implementation of other spatial operators have been also optimized on the PAR taking care of its constraints (4-connectivity). The 2D filters are split into 1D filters (figure 6). There are, at least, two passes: one pass for the vertical operator and one pass for the horizontal operator. After each pass, results are stored into memory. If the operator is not idempotent (like `ADD` used for the density computation) ($k \times 1$) and ($1 \times k$) operators are not split into smaller operators (figure 6, top). But if the operator is idempotent (like `AND` and `OR` operators used for ASF), each ($k \times 1$) and ($1 \times k$) operator is split into a set of (3×1) and (1×3) operators

(figure 6, bottom), with, at each time, a memory access. This decomposition reduces memory access to directly connected neighbors.

Thus, for $k > 3$, non idempotent $k \times k$ operators are expensive to implement. There are two reasons: the first one is, the great amount of cycles dedicated to gather far pixels to the current PE and the second one is the cost of *serial-bit ALU* operations. For these reasons the density filter is much more slower than the erosion/dilatation filter (respectively $\times 4.5$, $\times 5.9$ and $\times 6.3$ slower). As ASF are based on erosions and dilatations, their implementation remains efficient even if they have a great complexity, making them faster than density operator.

2.2.3 Hybrid algorithms

Although *Pvlsar34* can simply and quickly compute relaxation operators such as skeletons, or morphological connected operators, its efficiency in terms of useful computation is low for such irregular operators because of the expense due to the synchronous sequencing of the whole grid, that will only serve in some specific regions of the image. For that reason, a reduced set of asynchronous operators has recently been proposed by (22) to increase the computing level of the PARs. Thus, programmable connections, spanning tree constructions, `OR` and `SUM` asynchronous associations will be integrated in the next generations of PARs.

Such hybrid synchronous/asynchronous architecture will allow us to perform operations over a selected region (connected component) very efficient. This is the case of the geodesic reconstruction, which is useful for the motion detection algorithm (see Section 1.2). In the asynchronous model, the corresponding operation is computed like in the Associative Mesh (see Section 2.3): the reference set X is used as a binary mask to open/close the connections of the programmable mesh. Then an `OR-association` is computed on the marker set Y; the output is the result of the reconstruction $Rec^X(Y)$.

2.3 Associative Mesh

The Associative Mesh (17) intends to exploit a massive data-parallelism, originating from a model based on network reconfigurability: the Associative Nets Model (31). To allow efficient hardware optimizations for the large diversity of algorithms in image processing (32), the architecture is built from the observation of data-movements and data-structures encountered in this field. The Associative Mesh relies on a dynamic reconfigurability of its processors network and on an asynchronous electronic used to perform global operations and communication tasks. Reconfigurability and asynchronism offer solutions to adapt architectures to this context (5). Several studies have shown that most techniques of image processing can be implemented using the Associative Mesh (14) (15) (7) (3) or architecture using some of the implementation techniques of the Mesh and the Associatives Nets concepts (20) (21).

2.3.1 Associative Nets Model Theory

The Associative Nets Model is characterized by the application of associative operators on a locally reconfigurable, directed interconnection graph called *mgraph* implemented locally in each processor to enable its dynamic evolution in the course of an algorithm. *Mgraphs* can represent objects (figure 7) coded, processed or manipulated in image processing such as connected areas, edges, oriented trees... It allows us to think not only in terms of point-to-point communication between processors but to apprehend information at a higher level.

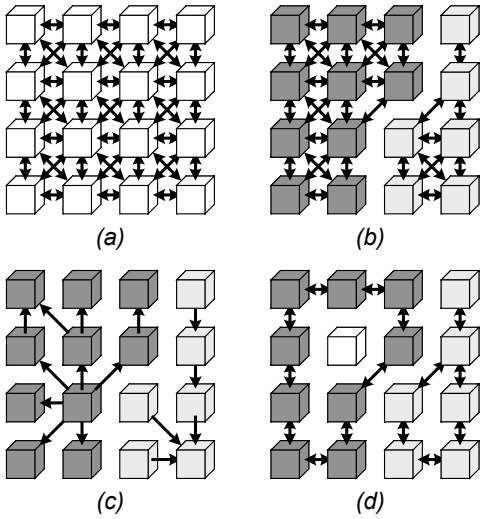

Fig. 7 Different mgraphs configurations:
(a) Full graph (b) Region graph (c) Oriented trees (d) Edges graph

Operations in the Associative Nets Model combine communication and computation aspects and are called 'associations'. They consist in a global application of an operator - such as logical operators, addition, minimum/maximum or spanning tree generation - on data spread over a connect set of the considered *mgraph*. As a basic example, this primitive can be used to asynchronously compute the area of a region by globally summing 1 per pixel on the mgraph connected components. It happens that most complex algorithms can be realized by iterating these primitive operations. Local associations are also allowed and are named *Step Associations*; the operator in this case is used to combine the local value of a processor with its nearest neighbors on the *mgraph*. Figure 8 presents an example of a global `MAX-Association`.

2.3.2 Associative Mesh Architecture

The Associative Mesh is a SIMD hardware transposition of the Associative Nets Model, featuring an 8-connected 2D mesh. Its originality comes from an asynchronous implementation of associations: the interconnection graph can be seen as an asynchronous path where data freely circulate

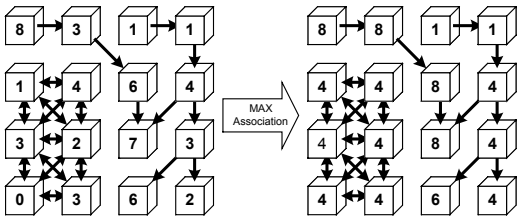

Fig. 8 MAX-Association

Affectation
`Equal`
Boolean Instructions
`OR, AND, XOR, NOT, Shift`
Comparisons
$=, \neq, >, \geq, <, \leq$
Arithmetical Operations
$+, -, \times, /,$ `modulo`

Table 4 List of SIMD instructions on the Associative Mesh

from a processor to another, propagating local results to neighbors until global stability is reached. Reconfigurability directly stems from the concept of mgraphs: each processor includes an 8-bit mgraph register, where each bit emulates the absence or presence of an incoming edge originating from a neighboring processor. The mgraph register is connected to the input of an `AND`-gate mask, which filters data emitted by the neighbors.

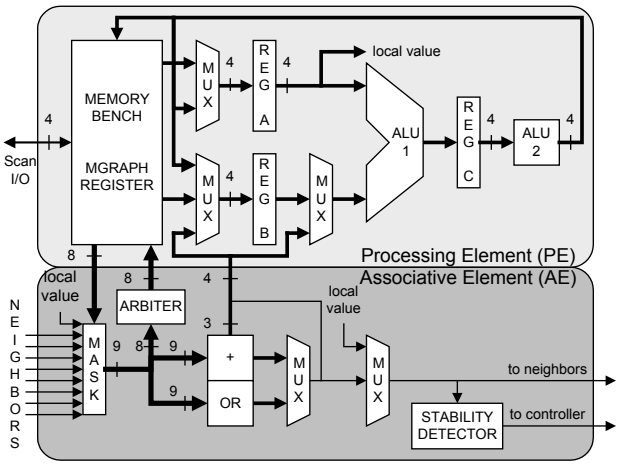

Fig. 9 Processor Architecture

A Mesh processor is built around two distinct parts: an Associative Element (AE) which performs asynchronous associations and a Processing Element (PE) dedicated to internal operations and memory tasks, featuring an all-purpose memory bench, dedicated registers to save the local mgraph value, an independent scan-register for image input/output and an ALU to perform basic local operations (figure 9).

In order to save space, AEs have a 1-bit data-path. A n-bit association will then be performed as an iteration of 1-bit associations. Operators have been designed to ensure that data cross a minimum of logical layers to optimize the traversal time of each AE. As an example, in a simulation based on a 90 nm technology and a Mesh running at 500 MHz, 40 AEs can be crossed in one clock cycle during an `OR-Association`. As a result, the basic global primitives of the model, associations, are performed in a very interesting computation time: simulations using the same technological parameter indicate that for a 512×512 image, `OR-Association` on 8-bit data is performed in 60 ns and `PLUS-Associations` in 200 ns (11). Such a speed on global operations emphasizes the impact of the asynchronous network on the Mesh's performance. Available instructions are listed in Table 4.

2.3.3 Processor virtualization & SIMD

With current technologies, the architecture discussed above is not optimized with a SoC approach, meaning a complete image analysis machine inside one chip. We can improve the Mesh integration by changing the PEs granularity: We now assign a group of N pixels to each processing element (now called SIMD PE) and consider that we have N virtual PEs per physical SIMD PE (N is called degree of virtualization) (10). To retain the benefits of asynchronism (very fast computation time, easy controllability), the AE structure is preserved in its original configuration. Thus, only the synchronous parts of the design are affected by the virtualization process. Figure 10 presents a virtualized PE dealing with two pixels.

This reorganization allows us to envision the architecture as the juxtaposition of an asynchronous communication network and a set of virtualized synchronous units, each managing N pixels. This new structure enables a significant area gain: we have shown that the design area is reduced by 20% if N=16, 25% if N=1024. With N=1024, the hardware cost of a 256×256 Associative Mesh, including 64 SIMD PEs, each managing 32×32 pixels, is about 165 millions of transistors. However, virtualization induces an increase of computation time due to the serialization of local operations. Still, this increase can be limited by implementing a SIMD unit in each SIMD PE, so we can parallelize, up to a certain point, operations for pixels managed by the same SIMD PE and reduce computation times in significant proportions (11).

2.3.4 $\Sigma\Delta$ initialization

The $\Sigma\Delta$ initialization is entirely performed by the SIMD PEs. Parallel conditional statements like `WHERE` or `ELSE WHERE` implement the `IF-THEN-ELSE` instructions by performing a sequence of operations in each PE, according to the result of a local logic comparison.

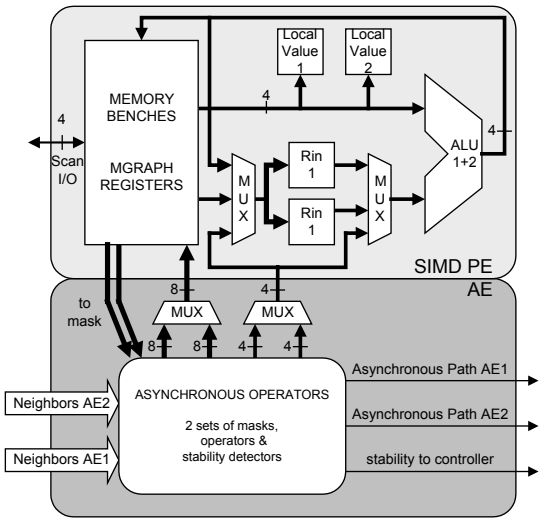

Fig. 10 Processor Architecture with virtualization

2.3.5 Markov

The update strategy used is *image recursive* for full-parallel updates. The energies computation are held by the SIMD PEs while the AEs are used to compute p, the sum of spatial 8-connected sites, using a local PLUS−ASSOCIATION. Conditional statements are used to collect sites label from E_{t-1} and E_{t+1}, compute V_p and V_f and also to set the final label to the site, depending on the total energy u. Note that the graph mask is configured by the set of the 8 masks (the four principal directions: North, South, West, East and the four secondary directions North-West, North-East, South-West and South-East).

```
// Difference of Adequacy energy computation
ΔU_a = (α × (2 × o × α))/4 × σ^2;
// Energy due to potential V_p
WHERE(E_{t-1} == 1)   U_p = −β_p;
ELSEWHERE U_p = β_p; ENDWHERE;
// Energy due to potential V_f
WHERE(E_{t+1} == 1)   U_f = −β_f;
ELSEWHERE U_f = β_f; ENDWHERE;
// Energy due to potential V_s
// Graph configuration: fully 8-connected graph
Graph = mNW+mW+mSW+mS+mSE+mE+mNE+mN;
s=PLUS-ASSOCIATION(Graph),E_t);
U_s = (8 − 2s) × β_s;
// Model energy computation
U_{m1} = U_s + U_p + U_f;
// Pixel labeling
WHERE(U_{m1} < ΔU_a) Ê_t = 1;
ELSEWHERE Ê_t = 0; ENDWHERE;
```

Associative Mesh ICM version

2.3.6 Binary geodesic reconstruction

Reconstruction takes an efficient use of the Mesh's AE units: the geodesic mask is represented as a graph, where each object is a unique connected component. Pixels of the mask (set to 1) are linked together with the LINK−WITH−ONES mgraph creation primitive. The markers are then dilated up to the mask's limits by performing a global OR−ASSOCIATION on the graph. The worst case is met when a unique object - shaped as a spiral - with a marker on one of its extremities fills the 200×200 image. Simulations based on a 90 nm technology reveal that for this extremely rare configuration, this operation on a Mesh running at 500 MHz will take 500 cycles. However, since data are 1-bit wide, it will only take a handful of cycles in most cases for the association to complete, thus providing a very interesting computation time.

```
// Creation of a graph representing the geodesic mask
// from the ImageMask binary image
LINK-WITH-ONES(GraphMask,ImageMask);
// Markers dilatation
Result=OR-ASSOCIATION(GraphMask,ImageMarker);
```

Binary geodesic reconstruction on Associative Mesh

2.3.7 Morphological opening

A dilatation on binary data, with a 3×3 structuring object, is simply achieved by a local OR−ASSOCIATION. Operating with a 5×5 or 7×7 structuring object only requires an iteration of local associations. Erosion is computed in a similar way, this time with an AND−ASSOCIATION. Therefore, a morphological opening will be implemented on the Mesh by computing 1, 2 or 3 local AND−ASSOCIATION, followed by 1, 2 or 3 OR−ASSOCIATION, depending on the size of the object.

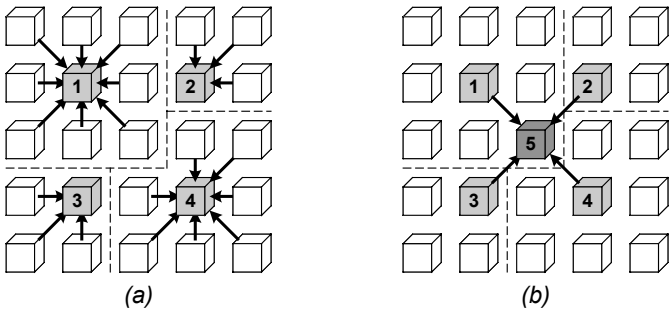

Fig. 11 (a): Sub-window split for a 5×5 density operation
(b): Sub-totals propagation to the central node

2.3.8 Density operator

On a 3×3 window, each pixel's 8 neighbors are summed in parallel by a local PLUS−ASSOCIATION. The final threshold is performed in the SIMD PEs. To operate on larger windows, we must ensure that each pixel will be counted once and once only. Addition is not idempotent, so it is impossible to simply iterate local associations as we did with the morphological opening. In consequence, we have to divide a 5×5 or 7×7 window into 3×3 or smaller sub-windows (figure 11). A sub-total is then computed in each sub-window,

using a local PLUS−ASSOCIATION. Finally, each sub-total is sequentially propagated to the central node to be added in the final sum. A last threshold is then performed in the SIMD PEs. As Boolean associations like OR−ASSOCIATION and AND−ASSOCIATION are idempotent, they require less graph configurations and associative operations. For the opening, respectively 2 and 3 for 5×5 and 7×7 associative operators and only 1 graph configuration versus 5 and 17 associative operators and as many as graphs configurations for non idempotent operation like the addition for the density operator.

```
// Graph config and sub-total for the upper left 3x3 window (1)
// (for example, the mW value activates the link between
// the processor and its west neighbor)
Graph=mNW+mW+mSW+mS+mSE+mE+mNE+mN;
Sum1=OR-LOCAL-ASSOCIATION(Graph,Image);
// Graph config and sub-total for the upper right 2x2 window (2)
Graph=mE+mNE+mN;
Sum2=OR-LOCAL-ASSOCIATION(Graph,Image);
// Graph config and sub-total for the bottom left 2x2 window (3)
Graph=mW+mSW+mS;
Sum3=OR-LOCAL-ASSOCIATION(Graph,Image);
// Graph config & sub-total for the last sub-window (4)
Graph=mW+mSW+mS+mSE+mE+mNE+mN;
Sum4=OR-LOCAL-ASSOCIATION(Graph,Image);

// Propagation to the central node
Graph=mNW; // for sub-window (1)
Final=OR-LOCAL-ASSOCIATION(Graph,Sum1);
Graph=mNE; // for sub-window (2)
Final=Final+OR-LOCAL-ASSOCIATION(Graph,Sum2);
Graph=mSW; // for sub-window (3)
Final=Final+OR-LOCAL-ASSOCIATION(Graph,Sum3);
Graph=mSE; // for sub-window (4)
Final=Final+OR-LOCAL-ASSOCIATION(mSE,Sum4);

// Final Threshold
WHERE(Final>12) Pixel=1;
ELSEWHERE Pixel=0;
ENDWHERE;
```

5×5 density operation on Associative Mesh

2.4 Architectures specification summary

In order to compare the three architectures and to focus on their advantage and drawback, their specifications are summed up into two tables: the architectural specifications (RAM, amount of transistors and power consumption) and bandwidth specifications (access to internal data and external data).

The table 5 provides the size of the internal RAM (size of the cache hierarchy on PowerPC G4 and size of the distributed memory on the Mesh and the retina) and an estimation of the power consumption. For the PowerPC G4, this is an average value, for the Mesh this is an estimation and for the retina, this is the measured value.

One very important point for comparison is the bandwidth of these architectures. As the Mesh and Retina are parallel architecture, we use the concept of *aggregate bandwidth* originating from High Performance Computing. The aggregate bandwidth is the sum of the bandwidth of all processors (table 6). Then we consider the internal bandwidth

archi	freq	internal RAM	transistors	Watt
AM	500 MHz	2 MB	160 M	2 W
Retina	5 MHz	225 KB	4 M	100 mW
G4	1 GHz	32KB + 512 KB	58 M	10 W

Table 5 architectures specifications

architecture	external bus	internal bus
AM	64 B/c	1024 B/c
AM	30 GB/s	476 GB/s
Retina	555 B/c	833 B/c
Retina	2.8 GB/s	4.1 GB/s
G4	1 B/c	16 B/c
G4	1 GB/s	16 GB/s

Table 6 Bandwidths: per cycle and per second

as the bandwidth between the processor and its closest RAM (L1 cache for the PowerPC G4 and distributed internal RAM for Mesh and Retina) and the external bandwidth as the bandwidth of the external bus, connecting the processor, to the external RAM or to another processor. For the Mesh, this is the capability of the asynchronous network to transfer data from one AE to another AE. For the Retina this is the bandwidth to transfer data from one memory bank associated to one processor to one of its connected processors. The reason is that, for the retina, the bandwidth cannot be computed in the same way than for RISC processor or an associative network, where internal and external buses can be easily identified. Each elementary processor (PE) of the retina has 48 bits of memory and 4 bits are shared with the four neighbors. Internal bus bandwidth capacity is based on the number of cycles for a READ, that is 6 cycles. External bus capacity is the number of cycle to perform a copy from one of the four bits (6 cycles for the READ) to one of the 44 private bits (3 cycles for the WRITE). Internal bus bandwidth is to access private memory, external bus to access shared memory. Note that for the Mesh, the bandwidths are computed for an architecture of 256×256 AEs with a virtualization N of 1024, that is 64 SIMD PEs.

We can notice and it is one of the main advantage of specialized architectures, that both Retina and Mesh can transfer much more data per cycle than a generalist RISC processor ($\times 64$ for internal and external buses). When considering bandwidth per second, the total aggregate bandwidth of the Mesh is close to the latest Cray vector processor performance (9) which has a peak bandwidth of 800 GB/s. Keeping in mind that most of the image processing algorithms are faced with *memory wall problem*, it is like if RISC still *wait for data* when the distributed buses of specialized machine can transfer data in time to feed processors.

3 Benchmarks

In order to compare the architectures, both from a qualitative and quantitative point of view, we used the frame rate and the *cpp* (*Cycle Per Point*):

$$cpp = \frac{t \times F}{n^2}$$

Where t is the execution time, F the processor frequency and n^2 the number of pixel to process, per processor. The cpp is an architectural metric to estimate the adequacy of an algorithm to an architecture (25). For each architecture, we provide the cpp and the speedup for every operator ($\Sigma\Delta$, ICM, morphological operator) and also for the whole algorithm as described in the first section. The algorithms have been implemented on a PowerPC G4 and PAR and have been simulated on the Associative Mesh with SystemC. For parallel architectures, the cpp expression is modified, depending on the number of pixels to be processed by a processor:

$$cpp_{PAR} = t \times F, \quad cpp_{Mesh} = \frac{t \times F}{n^2/N}$$

The cpp values have been calculated for 128×128, 256×256, 512×512 and 1024×1024 image size to analyze the cache behavior. We only provide the results for 256×256 image size to reduce the amount of results. For specialized parallel architecture like PAR or Mesh, the scalability is quite ideal so extensive results will not provide more information. For the PowerPC, more detailed results are provided to focus on the problem of cache misses.

3.1 Benchmark procedure

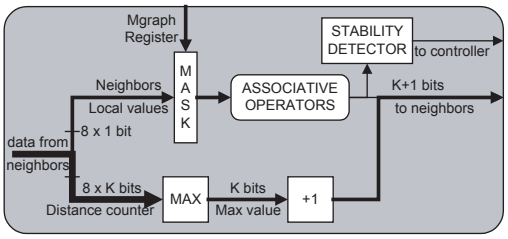

Fig. 12 Asynchronous data path on Associative Mesh simulator

For the PowerPC, we used the following approach. As there is no clock cycle 64-bit counter, on powerPC under MacOS, we have used a micro-second counter based on the micro kernel MACH. As execution time is very short for small images, the measure is done on i iterations of the loop, to get a duration of $\times 1000$ the resolution of the timer. As this measure can be polluted by the OS, r runs are performed, and the minimum is selected.
For the PAR, a logic analyzer Agilent 1670 has been used. Acquisition, conversion and computation time are readable on the analyzer. The figures only take into account the computation time.
For the Mesh, algorithms have been implemented and simulated using a Mesh simulator based on a SystemC description of the architecture allowing a cycle-accurate evaluation. In order to achieve this feature, we need to evaluate the duration of an association. Besides technological or architectural issues, this duration depends on the initial value of the data and of the graph type used for the operation. For instance, an OR-based-association computation time is given by the longest distance between two logical 1s, Therefore, estimation can be performed by computing the number of processors walked through by data during the operation. To implement this process on the Mesh simulator, data circulating in the asynchronous network provide two informations, each going through a specific data path: on one hand, the local result of the association as a 1-bit value uses the standard architecture data path and, on the other hand, a counter representing the number of processors walked through so far by this data, which is incremented after going through a processor (figure 12). When the association terminates, data in the network with the highest counter value gives the duration of the association.

3.2 PowerPC G4 results

Four algorithms have been benchmarked: ICM, $\Sigma\Delta$, density filters for 3×3 and 7×7 kernels and also Frame Difference (FD) algorithm. We added FD to get a reference in term of complexity and then in term of cpp, since no algorithm can be simpler than an absolute difference followed by a threshold to detect motion.

For each algorithm, two scalar versions and two "vector" versions were coded:

- s_0: scalar with no optimization, straight-forward coding,
- s_1: scalar with all possible optimizations,
- v_0: SWAR vector version with no optimization,
- v_1: SWAR with optimization like Loop unrolling, Register Rotation, strength reduction and computation factorization.

We provide cpp for classical image size, to point out the problem of cache behavior. For each algorithm, four ratios are also calculated:

- s_0/s_1: the impact of scalar optimization,
- v_0/v_1: the impact of SWAR optimization,
- s_1/v_1: the impact of SWAR switch for both optimized versions,
- s_0/v_1: the total acceleration from a basic/naive code to an optimized SWAR code.

We can see that the global speedup ($s0/v1$) is huge: from $\times 17$ for $\Sigma\Delta$ to $\times 60$ for 7×7 density filter. We can notice too that the code vectorization is the optimization technique that provides the highest speedup(line s_1/v_1: from $\times 6.8$ for ICM to $\times 15.6$ for density filter, while the scalar techniques all together provide a speedup (s_0/s_1) from $\times 1.2$ for $\Sigma\Delta$ to $\times 6.6$ for ICM. Such value of speedups make the use of optimization and vectorization to assert themselves for real-time computing on generalist purpose SWAR RISC processor.

algo	FD	$\Sigma\Delta$	ICM	Density3	Density7
cpp of scalar and vector versions					
s_0	28.7	50.5	112.2	27.2	114.0
s_1	17.5	40.5	16.9	12.3	30.1
v_0	3.4	4.5	3.4	1.9	5.7
v_1	1.2	3.0	2.5	1.5	2.5
gain between scalar and vector versions					
s_0/s_1	$\times 1.6$	$\times 1.2$	$\times 6.6$	$\times 2.2$	$\times 3.9$
v_0/v_1	$\times 2.9$	$\times 1.5$	$\times 1.4$	$\times 1.2$	$\times 2.7$
s_1/v_1	$\times 15.0$	$\times 13.7$	$\times 6.8$	$\times 8.2$	$\times 15.6$
s_0/v_1	$\times 24.6$	$\times 17.1$	$\times 44.9$	$\times 18.3$	$\times 60.0$

Table 7 implementation on PowerPC G4: *cpp* and gain

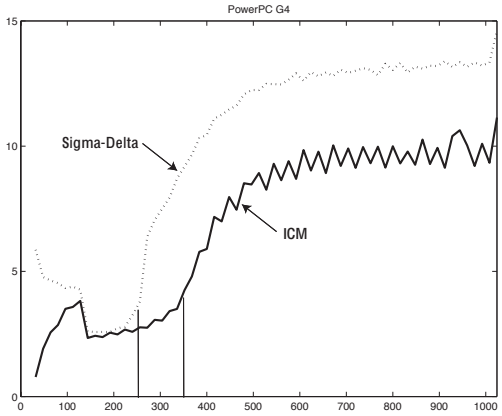

Fig. 13 ICM and $\Sigma\Delta$ *cpp* on PowerPC G4

version	*cpp*	gain
basic	147	-
LUT	41	$\times 3.6$
internal Loop Unrolling	32	$\times 1.3$
external Loop Unrolling	20	$\times 1.6$
SIMD vectorization	2.6	$\times 7.7$

Table 8 PowerPC G4 optimizations impact for ICM

Function	#i	#c	t(ms)
acquisition	0	0	15
digital conversion	64	2.5k	2
$\Sigma\Delta$ estimation	160	6.5k	1.3
spatial binary morphology:			
3×3 ASF	25	1k	0.2
5×5 ASF	58	2.3k	0.5
7×7 ASF	108	4.3k	0.9
3×3 density operator	36	2.2k	0.4
5×5 density operator	95	6k	1.2
7×7 density operator	152	9.6k	2.0

Table 9 Computation costs of the different algorithmic functions of $\Sigma\Delta$ detection on *Pvlsar34*.

Note that all the versions s_1, v_0, v_1 require some expertise from the developer. If these versions have been compiled with all optimization options of the compiler, without a little help, the compiler can not achieve a level of performance higher than the s_0 version.

If we look in detail at the figure 13 that represents the *cpp*'s evolution of ICM and $\Sigma\Delta$, for image sizes varying from 128×128 to 1024×1024, we can focus on two points. First there is a big gap in performance when image size increases and data do not fit in the cache. This phenomenon appears for different image size, depending on the algorithm (about 250×250 for $\Sigma\Delta$ and 350×350 for ICM). Then if both *cpp* are similar in the left part of the figure for small image sizes, the $\Sigma\Delta$ *cpp* becomes 40% bigger than ICM *cpp*. The *cpp* value is multiplied by $\times 3.8$ between left and right part of the figure. This result is in contradiction with any complexity analysis: $\Sigma\Delta$ is more simple than ICM, but because it requires more images to be present at the same time in the cache and also because there is very few instruction to optimize, there is no possibility to optimize the code. The $\Sigma\Delta$ algorithm is a typical case of *memory bounded problem*. The performance decrease is more important than for ICM or other algorithms studied here. That raises another problem: *SIMDization* is efficient only when data fit in the cache: if the global speedup (s_0/v_1) is $\times 17.1$ for 256×256 images size, it is only $\times 5.2$ for 1024×1024 images size.

Finally, if we compare the *cpp* of the best version (v_1) of ICM or $\Sigma\Delta$ algorithm with the naive scalar version (s_0) of FD or even the optimized scalar version (s_1), we can see that the *SIMDization* makes complex algorithms like Markov Random Field relaxation, or $\Sigma\Delta$ filtering run faster than FD. From a qualitative point of view, this enforces the use of SWAR on general purpose RISC computer since such SIMD multimedia instructions make robust algorithm run faster than naive algorithm if this one is not optimized.

In the next subsection we will focus on the implementation of these algorithms on the Retina and on the Mesh to finally compare them from an *embedded point of view*: frame rate and power consumption.

Table 8 shows, for PowerPC G4, that scalar optimizations are as important as SWAR optimizations: $\times 7$! As usual, the most efficient optimization is the highest level optimization: the algorithmic transform by LUT utilization provides a speedup of $\times 3.6$. Caches have also an important impact on performance whether the data fit in the cache (256×256) or not (512×512 and more).

3.3 Retina benchmarks

The results presented in this section, related to computation time and energy, have been measured on our experimental device composed of the 200×200 PAR *Pvlsar34* connected to an *Excalibur* board EPXA1, used to control the PAR and to perform higher level computations. The measures have been made using an oscilloscope and a logic analyzer except for density operator where figures are estimated, not measured.

Table 9 details the cost in time of the different functions. The first column represents the number of Boolean instructions, the second column the number of clock cycles and the third column the time, in ms. The acquisition corresponds to

Algorithm	*cpp*	t(μs)	frame rate
Frame Difference	0.5	1.024	977×10^3
$\Sigma\Delta$	35	70	14.3×10^3
ICM	70	140	7.14×10^3
Geodesic reconstruction	0.46	0.94	1.07×10^6
3×3 Morphological opening	0.19	0.39	2.58×10^6
7×7 Morphological opening	0.44	0.91	1.11×10^6
3×3 Density operator	0.32	0.65	1.55×10^6
7×7 Density operator	10.21	20.9	47.8×10^3

Table 10 *cpp* and frame processing rate of Associative Mesh (for 200×200 images)

image size	PowerPC G4	Retina	Mesh
Configuration #1: $\Sigma\Delta$ + 4 ICM			
frame rate	1178	-	24800
real-time Freq (MHz)	21	-	0.504
energy (μJ)	8500	-	201.6
Configuration #2: $\Sigma\Delta$ + morpho			
frame rate	3436	667	184000
real-time Freq (kHz)	7300	188	68
energy (μJ)	2900	150	27.1

Table 11 benchmarks results for configurations #1 & #2

the time of photo-transduction, during which no operation is performed. This time, measured here in normal conditions of our laboratory, naturally varies according to the lighting conditions. For the following functions (digital conversion, $\Sigma\Delta$ estimation and spatial binary morphology) the computation time only depends on frequency of the retina. For the spatial processing (binary morphology), three different sizes are considered for the largest radius of the structuring element set used both by the alternated sequential filter and the density operator.

If we only consider the computation time, the overall time consumed by the PAR is approximately 3.5 ms per frame, among which 2 ms for the CNA and 1.5 ms for the algorithm itself. These times are measured for a control frequency of 5 MHz. This means that, if we discard the acquisition time (which can make sense for a PAR observing a strongly lighted scene, for which the 2 ms of the CNA are sufficient as acquisition time), then a frame rate of 285 images per second is attainable at 5 MHz. Conversely, if the frame rate of 25 images per second is sufficient, then the control frequency can be lowered to 440 kHz, thus reducing proportionally the computing power.

At 5 MHz, the computing power of the whole device (PAR + EPXA1 board) has been measured at less than 1 W, from which only 100 mW is consumed by the PAR circuit and its cortex controller (external micro controller, figure 5) and the rest by the EPXA1 board. This means that the computing power of the PAR based vision system can certainly be lowered significantly by developing specific controlling ASIC instead of using off-the-shelf development kit.

3.4 Associative Mesh results

The results provided in the following section were simulated using a 90 nm technology parameter. On the Associative Mesh, the ICM *cpp* is about 70 for one ICM relaxation (varying from 70 to 80, depending on the degrees of SIMD and virtualization) and 35 for $\Sigma\Delta$.

For the morphological operator, the Associative Mesh *cpp* is higher than PowerPC G4 *cpp* because of 1-bit implementation of `PLUS-ASSOCIATION`. But with SIMD distributed processing power, it has the higher frame processing rate, even with virtualization.

The Mesh achieves spectacular performance. The bandwidth offered by its internal busses allow the Mesh to achieve a frame processing rate of 24800 images/s. This number could however be impacted by the performances and/or synchronisation with the video sensor. Another physical limitation is the number of incident photon impact(s) on the associated sensor.

3.5 Synthesis benchmarks

We only take into account the computation time and discard the acquisition time, the conversion time, the transfer time, and the power consumption of these operations. We are aware that results are a bit unfavourable to the PAR as the acquisition and conversion is integrated into itself contrary to the Mesh and the PowerPC. Right now, there is no way to get better results so the synthesis benchmarks will focus on the computation time to evaluate architecture performance. Considering the power consumption of a sensor – which is about 500 mW for both acquisition and conversion – the simulation and execution times will change but one order of magnitude between the PAR, the Mesh and the PowerPC performances will still exist.

Two configurations of benchmarks have been done (Table 11):

#1 $\Sigma\Delta$ + Markovian relaxation (4 iterations of ICM),
#2 $\Sigma\Delta$ + morphological post-processing (geodesic reconstruction, 3×3 density or 3×3 ASF, depending on the architecture).

In the configuration #1, $\Sigma\Delta$ is considered as a pre-processing algorithm used to provide a better initialization for ICM than classical Frame Difference algorithm. In configuration #2, $\Sigma\Delta$ is a "stand alone" algorithm with a post processing step to remove the remaining noise. The choice of spatial regularization algorithm has been done to be coherent with the architecture capabilities, that is:

- geodesic reconstruction on the Associative Mesh, since it is the strongest algorithm, by far and its implementation is efficient on Associative Mesh (compared to the implementation of the other architectures)
- 3×3 ASF on the PAR, since ASF is the most efficient operator on the retina.

image size	128	256	512	1024
Configuration #1: $\Sigma\Delta$ + 4 ICM				
G4 / AM	$\times 36$	$\times 42$	$\times 139$	$\times 155$
Configuration #2: $\Sigma\Delta$ + morpho				
G4 / PAR	$\times 14.4$	$\times 18.0$	$\times 61.3$	$\times 72.1$
G4 / AM	$\times 89$	$\times 107$	$\times 329$	$\times 395$
PAR / AM	$\times 5.5$	$\times 5.5$	$\times 5.5$	$\times 5.5$

Table 12 Energy comparison for configurations #1 & #2

- 3×3 density on SWAR CPU, to get a complexity that is comparable to PAR complexity, keeping in mind that after optimizations, SWAR 5×5 and 7×7 operators are quite as fast as the 3×3 operator.

To assess the performance of the retina, we only take into account the processing time ($1.3 + 0.2 = 1.5$ ms) not the total time (acquisition + conversion + processing). The reason is that both acquisition and conversion times are unknown for the PowerPC G4 and the Associative Mesh. This leads to a frame rate of 667 images/s. The estimation of energy consumption is based on this assumption.

Table 12 presents the energy consumption of the three architectures for the configurations #1 and #2. We can notice that specialized architectures are by far more efficient than the general purpose processor - even with SWAR computation: performance ratios are all greater than $\times 10$. This also means that even a 50% error, about the estimation of PowerPC G4 power consumption, is definitively not a problem.

3.6 Benchmark analysis

Before concluding, we focus, for each architecture, on the impact of the optimizations and the efficiency of the implementation.

- RISC PowerPC G4
  - From a point of view of embedded system, Altivec is well-adapted to complex algorithm like ICM relaxation: the ratio with Associative Mesh is $\times 35.7$ for configuration #1 and $\times 88.6$ for configuration #2. That could lead people to redesign SIMD Mesh PE architecture with an Altivec-like SWAR Instruction Set Architecture. For example a sub-set with only integer and also with restriction within the cross-bar capabilities could be integrated on a FPGA.
  - For RISC, SWAR is very efficient, since a complex and robust algorithm like those proposed in the configuration 1 and 2, are running faster, after *SIMDization*, than naive Frame Difference.
  - Another point for fair comparison, is the cache size of a RISC. We can see that the G4 is efficient (*cpp* low) for size up to 300×300. This means that for smaller size, the G4 efficiency is underestimated, from an embedded point of view, since it will work fine with smaller cache. Not only we can apply a down-clocking frequency for its embedded version, but we can also reduce its cache (both will decrease power consumption).
  - Down-clocking for *System on Chip*: Altivec frequency could be as low as 10 MHz for both configurations and for 128×128 and 256×256 images.
- Retina
  - The cost of the *serial-bit ALU* is a problem for arithmetic operators. A 8-bit ALU would have a great impact on performance, but will also have a negative impact of size and power consumption of the retina. A material *full adder* may be a golden mean to have good arithmetic performance.
  - Asynchronous logic and graph manipulation is a *must have* for specialized architecture, not only for low level operations, but also and especially for middle level operations with irregular processing like the morphological reconstruction. Next generation of artificial retina should integrate such kind of silicon graph management.
- Associative Mesh
  - Computation results show that the Associative Mesh is well suited for both configurations. Each sequence of algorithms takes advantage of one of the Mesh's architectural characteristics. For configuration 1, the massively parallel resources easily handle the amount of computation required by the ICM relaxation. For configuration 2, the dynamic reconfiguration of the graph's structure allows to efficiently represent the objects, while the asynchronous implementation of global operations guarantees a fast processing of the geodesic reconstruction. In both cases, frame rates are quite spectacular.
  - A remarkable aspect of the algorithms implementation on the Associative Mesh, in contrast with PowerPC G4 (and to a lesser extent, with retina) is that computation time is quasi-independent of the images' size or the detected object's shape.
  - The major drawback is the hardware cost of the Mesh to process big images when compared to the other architectures. Still, vision SoC implementation of a 256×256 Associative Mesh is compliant with current technology and only requires 3 times more transistors than a PowerPC G4 for a $\times 20$ faster computation.
  - With such performance, reducing the clock frequency by a factor 10 could still allow to process more than two thousand 256×256 images per second with a power consumption under 1 Watt. The Associative Mesh could then be used in association with a HD camera on a SoC platform.

4 Conclusion: future architectures

We have presented the implementation of robust sets of operator for Motion Detection, based on Markov Random field, Sigma-Delta filtering and morphological operators like open-

ing, density and Alternate Sequential Filter. These algorithms have been used to emphasize the intrinsic qualities and drawbacks of these architectures (section 4.1) and then to envision the specification of future architectures, first with SWAR paradigm (section 4.2), second with FPGA based customization (section 4.3) and finaly with many-core reconfigurable processor (section 4.4).

4.1 Pros and cons of the three architectures

- SWAR is efficient for low level algorithm. Currently used in RISC processor and also present and customized in some SoCs.
- Asynchronous Associative Networks. This model of computation is extremely efficient for both power consumption and intermediate levels algorithms. It is efficient for power consumption because asynchronism mechanism. It is also interesting for speed since, in our case, up to 40 asynchronous associative operators can be executed during 1 synchronous cycle. It is also efficient for intermediate level of processing because associative networks can be reconfigured and then, an operator can be applied through a graph, to any connected components. Any kind of irregular and CPU intensive algorithms can be handled efficiently, like geodesic reconstruction, watershed segmentation and of course, connected component labeling.
- Retinas are very low power embedded architecture. For tight integration and an optimized connection between sensor and calculator, retinas outperform SoC and Vision SoC systems like FPGA+sensor. But, right now they are limited to regular processing. Integrating an associative network inside a retina will allow to use such a kind of machine for intermediate level algorithm. So a quite complete image processing chain could fit into a high parallel and versatile system.

4.2 SWAR enhancement

Nowadays, the two main solutions to computer architecture limitations are: increasing RISC performance or customizing FPGA.
When RISCs have replaced CISCs using architectural optimizations like pipeline, registers and cache, the RISC motto was more instructions per cycle, because they were using less complex instructions that can be fetched, decoded and executed faster than CISCs can. As at this time, it was commonly accepted that clock frequency can go higher and higher. The easiest way, thanks to the technology, was to increase the clock frequency. At the same time two evolutions of RISC were released: the *superscalar* architecture (multi ALU/FPU per chip) and the *VLIW* (Very Long Instruction Word) like the Intel Itanium or the Texas Instrument C6x DSP family. But since a few years, clocks frequency does not increase as much as before. The new RISC motto could be “more instruction per second”. The General Purpose solution is the multicore approach (see section 4.4) And the Domain Specific solution is SWAR extension. As we can see in table 7, the speedup provided by Altivec – up to $\times 60$ – released in 1998 is by far, greater than the current number of cores inside a processor in 2008. As SWAR implementation requires *few* transistors because of the very simple control structure due to SIMD model, one very efficient way to increase performance could be “more SWAR into RISC”

- longer registers: 256 bits or even 512 bits, to process more data per cycle,
- more smart instructions: Altivec has a very useful *vector permutation unit* that provides powerful instructions like `vec_sel` that is an aesthetic way to perform a SIMD *if then else* condition (replacing masks computations and combinations) or `vec_perm` that can permute data with any kind of pattern (SSE can only do regular patterns of interlacing). `vec_perm` is used for computing unaligned vectors, matrix multiplications and even for sorting data. Such a kind of unit should be present in any SWAR architecture,
- more specialized or dedicated instructions like Altivec `vec_sum`, `vec_msum` that performs reduction into a register and the SSE2 `_mm_sad_epu8(a,b)` that performs a sum of absolute difference (SAD) between two registers. This instruction is used in every correlation algorithm based on block-matching. Adding such an instruction has been studied into (27).

4.3 FPGA based customization

Processor customization, as defined for reconfigurable architectures (37) and embedded systems, have to be explored. A customizable processor is a General Purpose Processor (GPP) embedded into a FPGA which cores can be enhanced. Most major FPGA manufacturers now provide solutions with softcore customizable FPGA (NIOS 2 for Altera, microblaze and PowerPC for Xilinx). Such technologies have room for improvements like adding new instructions, new customized format (35) for specific domain application (36) but also new dedicated blocks. With a compiler like C2H for Altera FPGA or DIME-C for Xilinx, a complete C function can be compiled into a VHDL block and be directly called inside a C code. GPP and its accelerators can then be seen as a full system on a chip. With these two levels of customization (instruction and hardware function) one can envision to add a new specialized instruction at the C level or new hardware function. A new instruction could be `b=sigmaDelta(a,b)` that compares `a` and `b` and increment/decrement `b` according to the result of the comparison. Such a function will remove *if then else* structure that stalls/flushes the pipeline. On the other hand an hardware function could implement a morphological op-

erator. Such hardware implementation can be much more faster than the sequential execution of the instructions that compose it, as no more "register to register" stage is required at each cycle like it is the case for pipeline execution. One of the best example of processor customization (not softcore but ASIP) is the Tensilica Xtensa architecture.

4.4 Many-core reconfigurable processor

If classic systems are able to race against Moore's Law (bigger caches, more complex branch predictors, more hardware optimizations), they are slowly but steadily losing the efficiency race. The amount of transistors involved in those systems keep increasing, but most of them are only use to stay on par with Moore's Prediction. They could be used more efficiently if GPP were no so "General Purpose" but also target some specific domains like computer vision or multimedia. The solution to this problem is brought by recent technology advances by combining both above solutions: designing reconfigurable parallel processor.
This leads to the fact that different models of computation have to be implemented in order to fit a given domain constraints. For example, the PAR can execute various regular algorithms in a very fast and efficient way even if the PAR itself is only composed of simple processors with few bits of memory. Similary, the Mesh, thanks to its asynchronous network can handle irregular algorithms. Another example of such a processor is the PIMM (23). PIMM is dedicated to morphological operations and use, an explicit hardware queue model to execute algorithms – like geodesic reconstruction or watershed segmentation – faster than a GPP, which are the most used architecture for such tasks but are, in fact, less efficient.
Currently, multi-cores approach is the leading solution for Thread Level Parallelism. But these processors are designed for regular processing, irregular processing still being out of their range. This also applies to CELL processor and GPU: Cell is *just* a 9-core heterogeneous RISC processor and GPU – from Nvidia or ATI – are still dedicated to regular processing even if they are going to be more flexible when used with a Stream Computing language – CUDA and Brook.
Next generation will include specialized/dedicated logic to tackle the problem of GPP inefficiency. Adding a custom part of logic (100 to 200 millions of transistors) is definitively no more a problem, compared to the total size of a CPU (800 millions of transistors for current Intel quad core). One of the most promising architecture of this kind is the Intel Polaris/Larrabee architecture from Terascale project (39). Polaris has a hierarchical bus to connect PEs together, PEs include an 512-bit SWAR unit and can be reconfigured, for 3D graphic processing or cryptography.

We believe that adding a custom part of logic into GPP (100 to 200 millions of transistors is now just a *part* of) dedicated to irregular processing would be a solution to the problem of GPP inefficiency.

Acknowlegment
We wish to thanks Joel Falcou for his valuable help.

References

1. R. Azencott. "Simulated annealing: parallelization techniques". John Wiley, 1992.
2. A. Bellon, J.-P. Derutin, F. Heitz, Y. Ricquebourg, "Real-time collision avoidance at road-crossings on board the Prometheus-ProLab 2 vehicle", Intelligent Vehicles. 1994.
3. A.Biancardi, M.Segrovia-Martinez, "Adaptative Segmentation of MR Axial Brain Images Using Connected Components", International Workshop on Visual Form, pp 295-302, Capri, Italy, 2001.
4. P. Bouthémy, P. Lalande. "Recovery of moving object in an image sequence using local spatiotemporal contextual information". Optical Engineering, 36-2, 1993.
5. G. Blelloch, "Vector Models for Data-Parallel Computing", MIT Press, Cambridge, 1990.
6. A. Caplier, L. Bonnaud, J.-M. Chassery, "Robust fast extraction of video objects combining frame differences and adaptive reference image", International Conference on Image Processing, 2001.
7. M. Ceccarelli, A. Petrosino, "A parallel fuzzy scale-space approach to the unsupervised texture separation", Pattern Recognition Letters, Vol 23, 5, pp 557-567, March 2002.
8. F. S. Cohen, D. B. Cooper, "Simple parallel hierarchical and relaxation algorithms for segmenting non-causal Markovian random fields". IEEE PAMI, volume 9, 2, pp 195-219. 1987
9. Cray X1: http://www.cray.com/products/x1/specifications.html
10. J. Denoulet, A. Mérigot, "System on chip evolution of a SIMD architecture for image processing", Computer Architecture and Machine Perception, New Orleans, May 12-16, 2003. DOI: 10.1109/CAMP.2003.1598175.
11. J. Denoulet, A. Mérigot, "Evaluation of a SIMD architecture dedicated to image processing". Global Signal Processing, 2004.
12. J. Denoulet, G. Mostafaoui, L. Lacassagne, A. Mérigot, "Implementing motion Markov detection on General Purpose Processor and Associative Mesh". Computer Architecture and Machine Perception, pp 288-293, Palermo, Italy, 2005.
13. K. Diefendorff, P.K. Dubeyn R. Hochsprung, H. Scales, "Altivec extension to PowerPC accelerates media processing", IEEE Micro, March-April 2000.
14. B. Ducourthial, A. Merigot, "Parallel asynchronous computations for image analysis" vol 90, 7, Proceedings of the IEEE, pp 1218-1229, 2002.
15. B. Ducourthial, G. Constantinescu, A. Merigot, "Implementing image analysis with a graph-based parallel computing model" , Computing. Supplementum, GbR'97, Workshop on Graph based Representation, pp 111-121, Lyon, France, Paris 17-18, 1997.
16. P. Dudek, "An asynchronous cellular logic network for trigger-wave image processing on fine-grain massively parralel arrays", IEEE Transactions on Circuits and Systems - II:Express briefs, vol 53, 5, pp 354-358, 2006.
17. D. Dulac, A. Merigot, S. Mohammadi, "Associative meshes: A new parallel architecture for image analysis applications", Computer Architecture and Machine Perception, pp 393-399. News Orleans, USA, December 15-17, 1993.
18. R. Forchheimer and A. Astrøm, "Near-sensor image processing: a new paradigm", IEEE Transactions on Image Processing, vol. 3, pp 736-746, 1994.
19. W. Daniel Hillis, Lewis W. Tucker, "The CM-5 Connection Machine: a scalable supercomputer". Communications of the ACM, vol. 36, 11, pp 31-40, 1993.

20. B.Galilee, F.Mamalet, M.Renaudin, P.Y.Coulon, “Parallel Asynchronous Watershed Algorithm-Architecture”, IEEE Transactions on Parallel and Distributed Systems, vol.18, no.1, pp.44-56, January 2007.
21. V.Gies, T.Bernard, “Increasing Interconnection Network Connectivity for Reducing Operator Complexity in Asynchronous Vision Systems”, International Conference on Discrete Geometry for Computer Imagery, pp.1-10, Poitiers, France, April 2005.
22. V. Gies, T. M. Bernard: “Tree extension of micro-pipelines for mixed synchronous-asynchronous implementation of regional image computations”. Proceedings of SPIE, volume 5677, Sensors and Camera Systems for Scientific and Industrial Applications VI, SPIE 2005.
23. J-C. Klein, F. Lemonnier, M. Gauthier, R. Peyrard, “Hardware implementation of the watershed zone algorithm based on a hierarchical queue structure”, IEEE Workshop on Nonlinear Signal and Image Processing, Neos Marmaras, Halkidiki, Greece, 1995.
24. T. Komuro, I. Ishii, M. Ishikawa, A. Yoshida, “A digital vision chip specialized for high-speed target tracking”, IEEE Transactions on Electron Devices, volume 50, 1, pp 191-199, 2003.
25. L. Lacassagne, M. Milgram, P. Garda, “Motion detection, labeling, data association and tracking, in real-time on RISC computer”. International Conference on Image Analysis and Processing, pp 520-525, Venise, Italy, September 1999.
26. P. Lalande, P. Bouthemy, “A statistical approach to the detection and tracking of moving objects in an image sequence”. European Signal Processing Conference, vol 2, pp 947-950. 1990.
27. C. Limousin, J. Sebot, A. Vartanian, N. Drach, “Architecture optimization for multimedia application exploiting data and thread-level parallelism”, Journal of Systems Architecture: the EUROMICRO Journal, Volume 51 Issue 1, pp 15-27, january 2005.
28. F. Lohier, L. Lacassagne, P. Garda, “A New Methodology to Optimize DMA Data Caching: Application toward the Real-time Execution of an MRF-based Motion Detection Algorithm on a multiprocessor DSP”. International Conference on Signal Processing Applications and Technology, March 1999, Phoenix USA.
29. A. Manzanera, J. Richefeu “A robust and computationally efficient motion detection algorithm based on Sigma-Delta background estimation” Proceedings Indian Conference on Computer Vision, Graphics and Image Processing, Kolkata, India, December 2004.
30. A. Manzanera, J. Richefeu “A new motion detection algorithm based on Sigma-Delta background estimation” Pattern Recognition Letters, vol 28, 3, pp 320-328, February 2007.
31. A. Mérigot, “Associative Nets Model: a graph based parallel computing model”, IEEE Transaction on Computer, vol 46,5, 558-571, 1997.
32. A. Mérigot and B.Zavidovique, “Image analysis on massively parallel computers: an architectural point of view”. International journal of pattern recognition and image analysis. vol 6, 3, 2002.
33. P. Nadrag, A. Manzanera, and N. Burrus, “Smart retina as a contour-based visual interface”, ACM Distributed Smart Cameras Workshop, DSC’06, Boulder, USA, 2006
34. J. Sébot, N. Drach-Temam, “Memory Bandwidth: The True Bottleneck of SIMD Multimedia Performance on a Superscalar Processor”, Proceedings of the 5th International Euro-Par Conference on Parallel Processing, pp 439-447, Springer-Verlag, 2001.
35. S. Piskorski , L. Lacassagne, S. Bouaziz, D. Etiemble, “Customizing CPU instructions for embedded vision systems”, IEEE Computer Architecture, Machine Perception and Sensors, 2006.
36. S. Piskorski , L. Lacassagne, M. Kieffer, D. Etiemble , “Efficient floating point interval processing for embedded systems and applications”, International Symposium of Scientific computing, Computer Arithmetic and Validated Numerics, 2006.
37. List of reconfigurable architectures `http://www.site.uottawa.ca/~rabielmo/personal/rc.html`
38. A. Rodrguez-Vazquez, G. Linan-Cembrano, L. Carranza, E. Roca-Moreno, R. Carmona-Galan, F. Jimenez-Garrido, R. Dominguez-Castro, S. Espejo Meana, “ACE16k: The Third Generation of Mixed-Signal SIMD-CNN ACE Chips Toward VSoCs”, IEEE Transactions on Circuits and Systems, vol 51, 5, pp 851-863, 2004.
39. Intel Tera-scale project: `http://techresearch.intel.com/articles/Tera-Scale/1449.htm`
40. B. Zavidovique and G. Stamon, “Bilevel processing of multilevel images”, proc. PRIP’81, Dallas TX, Aug. 1981.