Implementing motion Markov detection on General Purpose Processor and Associative Mesh

J. Denoulet
Institut d'Electronique Fondamentale
Université Paris Sud
Email: denoulet@ief.u-psud.fr

G. Mostafaoui
Laboratoire des Instruments et Systemes
Université Paris 6
Email: mostafaoui@lis.jussieu.fr

L. Lacassagne and A. Mérigot
Institut d'Electronique Fondamentale
Université Paris Sud
Email: lacas|am@ief.u-psud.fr

***Abstract*— We present a robust implementation of a motion detection algorithm based on a markovian relaxation both on General Purpose Processors, and on a specialized architecture, the Associative Mesh. The Mesh architecture is an instance of the associative nets model targeting real time execution of low level image algorithms and vision-Soc implementation. The algorithm implementation on both architectures is described, and also the required optimizations to speedup the execution.**

Keywords

Vision-SoC, associative nets model, SIMD, motion detection, Markov Random Field.

I. Introduction

This paper presents the performance evaluation of the Associative Mesh architecture and SIMD PowerPC G4 for motion detection algorithm. Associative Mesh is a reconfigurable, asynchronous and massively parallel SIMD design, targeted towards image analysis implementation. Until now, most of the algorithmic studies suggest that, to reach maximal efficiency, the machine's physical topology should converge towards one SIMD processor per pixel. However, in the prospect of a visual-SoC implementation of the Associative Mesh, it is hazardous at this day to envision a 512×512 or even a 256×256 matrix of processors on one integrated circuit. A solution to achieve this project consists in changing the circuit's architectural organization by modifying the Mesh's granularity and using a smaller number of more complex processors (virtualization). In this paper, we present the new structure of the Associative Mesh and show the contribution of virtualization on the architecture's performance through the example of a motion detection algorithm based on markovian relaxation. The first section describes a robust enhancement of the classical motion detection algorithm, the second presents the Associative Mesh and its virtualization targeting SoC implementation. Third section deals with the optimizations required to speedup the execution of the algorithm on PowerPC and details the parallel and asynchronous implementation on the Associative Mesh. Benchmark results are given in the fourth section which provides information about bus bandwidth and "*memory wall problem*".

II. Motion Markov detection

Markov Random Field based algorithms have asserted themselves in a lot of image processing areas for regularizing ill-posed problems. Their main advantage is their robustness, and their well-known main drawback is their CPU consuming due to the huge amount of calculations. This has led researchers to study a lot of solutions to speed their execution up, as parallel machine, or dedicated architecture [1][2] [6] [14].

The latest version of the algorithm is composed of 3 parts:

- the *pre-processing step*: usually image difference and binarization to initialize the labels, now replaced by a $\Sigma - \Delta$ algorithm
- the *processing step*: a determinist relaxation (ICM) is applied to the image frame difference.
- the *post-processing step* (for color): an hysteresis threshold is applied to color images to strengthen the relaxation

A. Markov Random Field

The aim of the markovian process is to improve the quality of the image difference. Because of changes of illumination this image is very noisy: a lot of pixels are labelled as in motion and the regions are not filled. The energy model has been introduced by the LIS-Grenoble laboratory [5] and is derived from IRISA model [3][13].

Let I_t the grey level image at the moment t, and O_t the observation, that is the absolute difference between two consecutive images ($O_t = |I_t - I_{t-1}|$) or between the current image and a reference image ($O_t = |I_t - I_{ref}|$). Then thresholding O_t initializes the estimated field of labels $\hat{E}_t$. The Iterated Conditional Modes (ICM) is then applied to E_{t-2}, $\hat{E}_{t-1}$ and $\hat{E}_t$ to obtain the relaxed field E_{t-1}.

1) Energy clique function: The energy used u is the sum of two energies: u_m model energy that is a regulation term which ensures a spatio-temporal homogeneity and u_a adequation energy.

$$\begin{aligned} u(o_s, e_s) &= u_m(e_s) + u_a(o_s, e_s) \\ u_m(e_s) &= \sum_{c \in C} V_c(e_s, e_r) \\ u_a(o_s, e_s) &= \frac{1}{2\sigma^2}\left[o_s - \psi(e_s)\right]^2, \psi(e_s) = \begin{cases} 0 & \text{if background} \\ \alpha & \text{if motion} \end{cases} \end{aligned}$$

Clique associated to u_m energy is a spatiotemporal and second order clique V_c (figure 1) composed with a second order spatial clique associated to the potential V_s and two first order temporal associated to V_p et V_f. Potentials are defined as follow:

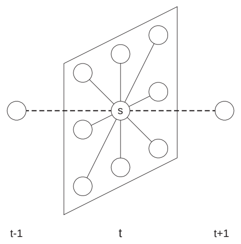

Fig. 1. spatio-temporal clique

$$V_c(e_s,e_r) = V_s(e_s,e_r) + V_p(e_s^t, e_s^{t-1}) + V_f(e_s^t, e_s^{t+1})$$

$$V_s(e_s,e_r) = \begin{cases} -\beta_s & \text{if} \quad e_s = e_r \\ +\beta_s & \text{if} \quad e_s \neq e_r \end{cases}$$

$$V_p(e_s^t, e_s^{t-1}) = \begin{cases} -\beta_p & \text{if} \quad e_s^t = e_s^{t-1} \\ +\beta_p & \text{if} \quad e_s^t \neq e_s^{t-1} \end{cases}$$

$$V_f(e_s^t, e_s^{t+1}) = \begin{cases} -\beta_f & \text{if} \quad e_s^t = e_s^{t+1} \\ +\beta_f & \text{if} \quad e_s^t \neq e_s^{t+1} \end{cases}$$

2) Sigma-Delta initialization: The use of a $\Sigma-\Delta$ algorithm designed by Manzanera [17] is an aesthetic way to tackle the problems of updating the reference image and automatic thresholding. It is composed of four steps: update the background image M_t with a $\Sigma-\Delta$ filter, compute the image of difference O_t, update the time-variance image V_t from O_t with a $\Sigma-\Delta$ filter and initialize $\hat{E}_t$. The typical value of N is in $[2..4]$.

```
for each pixel $x$:
  if $M_{t-1}(x) < I_t(x), M_t(x) = M_{t-1}(x) + 1$
  if $M_{t-1}(x) > I_t(x), M_t(x) = M_{t-1}(x) - 1$
  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 $V_{t-1}(x) < N \times O_t(x), V_t(x) = V_{t-1}(x) + 1$
  if $V_{t-1}(x) > N \times O_t(x), V_t(x) = V_{t-1}(x) - 1$
  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 $\hat{E}_t$

Another important robust feature of this $\Sigma-\Delta$ pre-processing is that it can be customized to different kind of motion and image noise, by specifying the update steps for the background image M_t, and the standard deviation image V_t.

We have modified the algorithm to enhance the detection quality: by updating M_t and V_t only if there is no motion in E_{t-1}. The M_t update becomes:

```
for each pixel $x$:
  if $E_{t-1}(x) = 0$ then
    if $M_{t-1}(x) < I_t(x), M_t(x) = M_{t-1}(x) + 1$
    if $M_{t-1}(x) > I_t(x), M_t(x) = M_{t-1}(x) - 1$
  otherwise $M_t(x) = M_{t-1}(x)$
```

step1': modified M_t update

3) Color version: For some very noisy sequences (bad conditions of acquisition, fast variations of illumination), the motion segmentation could be inefficient. A color version, which is more than a 3-copy version of the monochrome version, is a way to tackle such problems.

$$I_t = (I_t^R, I_t^G, I_t^B)$$

Three thresholds are required for the estimation of $\hat{E}_t$: θ_R, θ_G and θ_B. The pixel color represents the number of plane where motion is detected:

- primary colors (R, G and B): only one plan for motion,
- secondary colors, yellow ($R+G$) magenta ($R+B$) and cyan ($G+B$): two plans of 'motion',
- white ($R+G+B$): the pixel is estimated to be 'motion' on three plans.

In the second step, we apply the ICM for each plane independently of the others. The goal of this post-processing is to deal with *false detections*:

- pixel detected in motion, but corresponding to still background,
- pixel not detected in motion, and belonging to moving objects.

We developed three different post-processings:

- *level k diffusion*, applied after the 4 ICM iterations,
- *level k ceiling*, applied at each ICM iterations
- hysteresis threshold, applied after the 4 ICM iterations

Let (R, G, B) be the color components of a site s of $\hat{E}_t$ ($R, G, B \in \{0,1\}$). The definition of *diffusion* and *ceiling* are:

$$\text{ceiling-k}(s) = \begin{cases} (R,G,B) & \text{if } R+G+B \geq k \\ (0,0,0) & \text{if } R+G+B < k \end{cases}$$

$$\text{diffusion-k}(s) = \begin{cases} (1,1,1) & \text{if } R+G+B \geq k \\ (0,0,0) & \text{if } R+G+B < k \end{cases}$$

More higher is k, more the noise is removed from image, but more useful information (moving pixels) can be omitted. The ceiling method could be more robust than the diffusion method: by remaining the values unchanged, ceiling prevent the noise to propagate from on color channel to another.

Nevertheless, the dilemma "decrease the noise or recover useful information" still remains. A solution is to mix results of *high confidence* ($k = 3$) for very noisy areas with *low confidence* ($k = 1$) for slightly noisy areas. So an hysteresis threshold can be applied to the *number* of components of a site. Given two thresholds, (high & low) s_H s_L, a site s is:

- in motion, if $R+G+B \geq s_H$,
- stationary if $R+G+B < s_L$,

- in motion if $R+G+B \in [s_L..s_H]$ and if there is a connect path between the current site, and a site such $R+G+B \geq s_H$

Such efficient algorithm must be based on connected-component labelling algorithm where the tests are done once for each region and not for every pixel like the *LSL* algorithm [12][21]. The three possible couples of values for the thresholds are: (1,2), (1,3) and (2,3). To remove noise, s_H must be set to 3.

We have tested other color spaces like YUV or $YCbCr$, but they are too close to RGB for significant enhancement. HLS (*Hue Lightning Saturation*) is more robust, but the camera used to grab sequence had a small SNR and hadn't a tri-CCD sensor to prevent from chrominance decimation. Sequences are available at `www.ief.u-psud.fr∼lacas/demo/camp2005.html`

III. Associative Mesh

Algorithm implementation on a dedicated architecture often has a real-time execution aim. However, even though there are many architectures to perform low-level image processing, there is currently no machine able to efficiently perform a complete image analysis: the diversity of the implied algorithms complicates a unique representation of data-movements and their irregularity restricts architectural optimization options [20]. Reconfigurability and asynchronism offer solutions to adapt architectures to this context [4].

Our approach intends to exploit a massive data-parallelism, originating from a model based on network dynamic reconfigurability: the Associative Nets Model [18]. The Associative Mesh [10], based on this model, is a SIMD structure of processors built from the observation of data-movements and data-structures encountered in image processing, These movements are irregular, and changing according to data in the course of a process. The Associative Mesh's second originality comes from its asynchronous electronic used in communication and global operation aspects. This technique allows the design to save area, power and also reach a higher clock frequency in the synchronous part of the circuit. Several studies have shown that most techniques of image processing can be implemented using the Associative Mesh [11][19].

A. Associative Nets Model Theory

The Associative Nets Model is characterized by the application of associative operators on a locally reconfigurable directed interconnection graph $G = (P, E)$, where P is the set of nodes/processors and E, the set of edges joining processors in the graph. Operations are performed on a subset $G' = (P, E')$ of G, where E' is a subset of E. G' is also called *mgraph* and can be implemented by coding in every node a subset of the incoming edges of E. This local implementation by a parallel variable allows a dynamic evolution of *mgraphs* in the course of an algorithm.

Mgraphs can represent usual objects which are coded, processed or manipulated in an image such as connected areas, edges, oriented trees... Therefore, it will allow us not to think in terms of point-to-point communication between processors but rather apprehend information at a higher level, corresponding to manipulated objects.

OR-Association	60 ns
MAX-Association	80 ns
PLUS-Association	200 ns
Spanning Tree	20 ns

TABLE I
Association computation times for 512x512

Operations in the Associative Nets Model combine communication and computation aspects and are called associations. They consist in a global application of an operator - supposed to be associative and commutative - 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. Associations can also be used to compute the maximal - or minimal - value of a region, find its perimeter or its average value.

Operators used in associations include logical operators, maximum and minimum, addition, as well as *mgraph* manipulation primitives, in particular spanning tree generation. It happens that most complex algorithms can be realized by iterating these primitive global data movements. 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*.

B. Associative Mesh Architecture

The Associative Mesh is a SIMD architecture based on the Associative Nets Model, G being in this case a 8-connected 2D mesh. The structure's originality comes from an asynchronous implementation of associations, as there are no memory registers to stock local results before sending them to neighboring processors. Therefore, the interconnection graph can be seen as an asynchronous path where data freely circulate from a processor to another, and where local results propagate themselves to their neighbors until global stability is achieved. With this asynchronous treatment, the crossing time of each processor is very short and therefore the basic global primitives of the model, associations, are performed in a very interesting computation time (table I) for a frequency of 500 MHz.

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.

The elementary processor, also called PE, of the Associative Mesh is built around two distinct layers: an asynchronous layer which performs associations, and a synchronous layer dedicated to local operations and memory tasks, featuring an

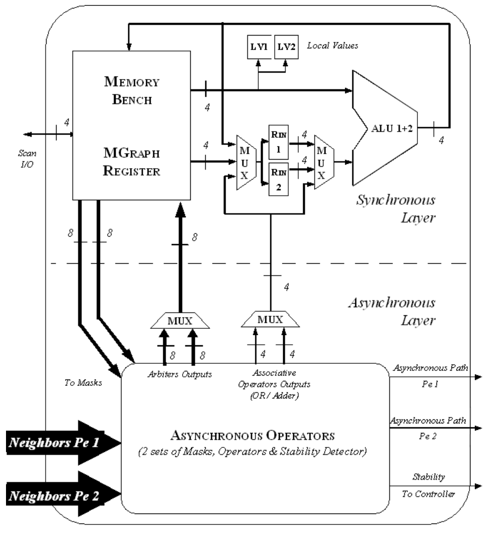

Fig. 2. Processor Architecture with virtualization

all-purpose memory bench, dedicated registers to save the local mgraph value, an independent scan-register for image input/output and a 4-bit wide ALU to perform basic local operations.

C. Processor virtualization & SIMD

With currently available microelectronic technologies, the architecture discussed above is not optimized with a SoC approach, meaning a complete image analysis machine on one chip. A way to improve the Associative Mesh integration is to change the structure's granularity by assigning a group of N pixels to each physical processor. So we now consider that we have N virtual PE per physical PE (we also call N degree of virtualization).

To retain the benefits of asynchronism (very fast computation time, easy controllability), the asynchronous layer is preserved in its original configuration. Thus, only the synchronous parts are affected by the virtualization process. Figure 2 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 of the image. This new structure enables a significant area gain: we have shown that the design's area is reduced by 20% if $N = 16$ and 25% if $N = 1024$ [8]. With $N = 1024$, the hardware cost of a 256×256 Associative Mesh, including 64 synchronous units managing 32×32 pixels, amounts to 165 millions of transistors.

A consequence of virtualization is an increase of computation time due to theserialization of local operations. But we can use the synchronous units to compensate for this cost: by implementing a SIMD unit in each virtualized PE, we will be able to parallelize, up to a certain point, local operations for pixels managed by the same virtualized unit, and reduce a significant amount of computation time [9].

IV. Software Optimizations

In order to perform a fair comparison of these architectures, the algorithm must be optimized for each one. Here are the set of optimisations (table II sums up all the impacts):

A. Remove tests & comparisons

The first step is to remove tests from potential V function. First, the sites are labeled $0/1$ rather than $-1/1$, then, the comparison of a site's state to the central site's state e_s is replaced by accumulation: we just take into account sites which state are 1 (p_1, s_1 and f_1 for the past, present and future images). With a central site state 1, the spatial energy is: $u_{m1} = (8 - 2s_1)\beta_s + (1 - 2p_1)\beta_p + (1 - 2f_1)\beta_f$, with $u_{m0} = -u_{m1}$. 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 is. This computation looks like a 3×3 recursive kernel filter: the output is one of the inputs of the next iteration. The u_m computation is simplified into the summation of pixels and, eventually, access to LUTs (*Look-Up Tables*) for multiplications.

B. Kernel optimization

The optimization of such kernel consists in unrolling the inner loop to take advantage of the overlapping windows. To go further, we must take into account the different update strategies:

- *site recursive*: each updated site is used for the current site relaxation
- *block recursive*: updated sites are used at the end of a block (or a line)
- *image recursive*: no use of updated sites for the current image relaxation

The site recursive strategy is the most constraining: there is a loop carried path dependency from two consecutive iterations, but it achieves the best energy minimization since all updates are used for each computation. The opposite strategy, the image recursive strategy, enables all kind of optimizations and the largest amount of parallelism to perform relaxations. This kind of strategy is implemented into massively parallel architecture [15][16].

C. Memory caches

Taking into account the processor memory hierarchy, we can reduce the duration of the relaxation by replacing four scans of the image with one relaxation per site by one scan with four relaxations per site. The figure 3 shows 3 optimized schemes for scanning an image. The distance between the sites depends on the strategy of both site update and neighbors update. On the left: optimal scheme for energy minimization: a simple 4-copy of the scalar version. In the middle a left-to-right order must be enforced to maximize the number of updated sites used. On the right the most parallel version: a line-recursive strategy with 4 lines computation. Only the right scheme easily allows to unroll the inner loop, while processing 4 lines in parallel, in a loop body.

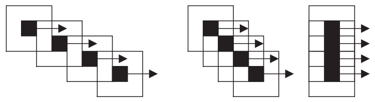

Fig. 3. scan strategies & optimized sites update

D. SIMD optimization

Except on former vector machines, SIMD sparse addressing (SIMD LUT access) is not implemented, so computations are done with arithmetic operations. With 128-bit registers, sixteen 8-bit computations can be done in parallel for the first steps of the algorithm (summations) and eight 16-bit computations for the energy computations. For u_a, a special fixed-radix operation is performed to implement $a{*}x/2^p$. Tests are handled through vector comparison. The $\Sigma - \Delta$ stage for M_t update is performed as follow:

```
if m < x then inc = +1;
if m > x then inc = -1;
if m = x then inc = 0;
m = m + inc;
```

scalar $\Sigma - \Delta$ version

```
inc1 = vec_cmplt(m, x);
inc2 = vec_cmplt(x, m);
m = vec_sub(m, inc1);
m = vec_add(m, inc2);
```

SIMD $\Sigma - \Delta$ Altivec version

E. Associative Mesh optimizations

The $\Sigma - \Delta$ initialization is done by the SIMD PEs. The update strategy used is *image recursive* for full-parallel updates. The energies computation are held by the SIMD PEs while the asynchronous network is used to compute p, the sum of spatial 8-connected sites, using a local PLUS-ASSOCIATION performed on the subgraph $G_1 = G$. Conditional statements like WHERE or ELSEWHERE are used to collect sites label from E_{t-1} and E_{t+1} to compute V_p and V_f and also to set the final label to the site, depending on the total energy u.

```
// Adequation energy computation
$U_a = M \times (2 \times \alpha)$;
$U_a = U_a - \alpha^2$;
$U_a = U_a / 4 \times \sigma^2$;
// Energy due to potential $V_p$
WHERE($E_{t-1} == 1$)    $U_p = -\beta_p$;
ELSEWHERE $U_p = \beta_p$;
ENDWHERE;
// Energy due to potential $V_f$
WHERE($E_{t+1} == 1$)    $U_f = -\beta_f$;
ELSEWHERE $U_f = \beta_f$;
ENDWHERE;
// Energy due to potential $V_s$
s=PLUS-STEP-ASSOCIATION($G_1$,$E_t$);
$U_s = (8 - 2s) \times \beta_s$;
// Model energy computation
$U_m = U_s + U_p + U_f$; // Pixel labelling
WHERE($2 \times U_m < U_a$)$\hat{E}_t = 1$;
ELSEWHERE $\hat{E}_t = 0$;
ENDWHERE;
```

Associative Mesh ICM version

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 II
POWERPC OPTIMIZATIONS IMPACT

V. BENCHMARKING

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

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

The *cpp* is an architectural metric to estimate the adequation of an algorithm to an architecture [12](PowerPC G4 specifications are given in table III). For each architecture, we provide, the *cpp* both for $\Sigma - \Delta$ and one ICM iteration, the *cpp* of the full algorithm and the frame rate (table V).

The algorithm has been implemented on a PowerPC G4, using Altivec SIMD instructions and simulated on the Associative Mesh with SystemC.

archi	freq	L1 / L2 cache	transistors	Watt
G4	1 GHz	32+32 KB / 512 KB	58 M	10 W

TABLE III
POWERPC G4 SPEC

If the Associative Mesh has an aggregate bandwidth of 400 GB/s, which is relatively close to the latest Cray vector processor whith a bandwidth of 800 GB/s [7], it has 64 up to 256 times more bandwidth than the generalist architectures which still *'wait for data'* when the Mesh's distributed buses can transfer 40 GB/s to feed processors (table IV).

architecture	external bus	internal bus
Associative Mesh	64	1024
G4	1	16

TABLE IV
BANDWIDTH: BYTES PER CYCLE

architecture	size	$\Sigma - \Delta$	ICM	cpp	t(μs)	rate
PowerPC	256	3.25	2.59	13.59	900	1123
PowerPC	512	12.25	8.87	46.11	12000	83
Associative Mesh	256	35	70	315	40.32	24801
Associative Mesh	512	35	70	315	40.32	24801

TABLE V
cpp AND FRAME RATE OF POWERPC G4, ASSOCIATIVE MESH

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 virutalization) and 35 for $\Sigma - \Delta$.

About cache impact: Sigma-Delta algorithm was developed for automatic pre-processing, with the conviction of a flyweight overhead (compared to frame difference and binarization). From an academic point of view its complexity is lower than the ICM complexity, whatever the implemented optimizations (even SIMD). But from a practical point of view, it is no more true: for large images, Sigma-Delta's *cpp* is multiplied by 3.8 between 256×256 and 512×512 images and is 40% bigger than ICM's one. Figure 4 shows that Sigma-Delta *cpp* is always higher than ICM *cpp* (for image size varying from 32 to 1024).

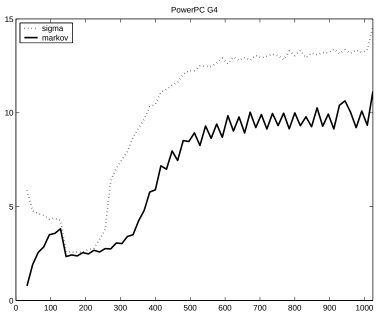

Fig. 4. ICM *cpp* on PowerPC G4

Table II shows, for PowerPC, that scalar optimizations are as important as SIMD 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 performances whether the data fit in the cache (256×256)or not (512×512 and more). The Associative Mesh *cpp* is higher than PowerPC *cpp* because of 1-bit implementation of `PLUS-ASSOCIATION`. But with SIMD distributed processing power, even with virtualization, the Mesh achieves spectacular performances. With a frame rate of 24801 images/s the processing limitation comes from the number of incident photon impact(s) on the associated sensor.

CONCLUSION

We presented the implementation of a robust motion detection algorithm based on markovian relaxation for two different architectures: PowerPC G4 and Associative Mesh. Algorithmic and software optimizations provide a speedup of $\times 56$ for the G4. The G4 implementation can lead to SoC integration: reducing the clock frequency from 1GHz to 100 MHz, for example, will always ensure real time execution (100 images/s) while the electrical consumption will be reduced to nearly 1 Watt.

We also showed the performance of a virtualized Associative Mesh. The implementation of such a CPU-intensive algorithm shows that complex treatments can be computed in a very interesting time. The adjunction of a SIMD ALU drastically reduces the processing latency introduced by virtualization. The objective of real-time image processing is still achived, even with operators primarily composed of local operations, which are yet the most penalizing for the virtualized architecture.

The area estimation allow us to envision a Visual-SoC implementation of an Associative Mesh with a large number of processors. With only three times more transistors the Mesh is twenty times faster, that is eight times more efficient!

Still, more architectural optimizations could be operated: for instance, with the current structure, an important number of redundant operations are performed by the synchronous units. Identical operations could be performed only once for each physical processor and then distributed to the virtual processors instead of being computed for each virtual processor. This would generate an additional speedup if we can manage the controllability issues raised by this modification .

REFERENCES

[1] R. Azencott. Simulated annealing: parallelization techniques. John Wiley, 1992.
[2] A. Bellon. Détection et suivi de véhicule en mouvement, implémentation parallèle sur un systèeme à mémoire distribuée. Thèse Lasmea 1996.
[3] P. Bouthémy, P. Lalande. Recovery of moving object in an image sequence using local spatiotemporal contextual information. Optical Engineering, 36-2, 1993.
[4] G. Blelloch, Vector Models for Data-Parallel Computing, MIT Press, Cambridge, 1990.
[5] A. Caplier. Modèles markoviens de détection de mouvements dans les squences d'images: approche spatio-temporelle et mise en oeuvre temps réel. Thse INPG 1995.
[6] J.P. Cocquerez, S. Philipp. Analyse d'images: filtrage et segmentation. MAsson 1995. Temps de calcul sur Sparc5 et CM2 page 220.
[7] Cray X1: http://www.cray.com/products/x1/specifications.html
[8] J. Denoulet, A. Mérigot, System on chip evolution of a SIMD architecture for image processing, CAMP 2003.
[9] J. Denoulet, A. Mérigot, Evaluation of a SIMD architecture dedicated to image processing. GSPx (Global Signal Processing) 2004.
[10] D. Dulac, Contribution au parallèlisme massif en analyse d'image: une architecture SIMD fondée sur la reconfigurabilité et l'asynchronisme. PhD thesis, University of Paris XI, 1996.
[11] S. Guezguez, Etude de l'adequation du modèle de réseaux associatifs avec les algorithmes d'analyse d'images.Thèse Université de Paris Sud, 1999.
[12] L. Lacassagne. Détection de mouvement et suivi d'objets en temps réel, Thèse Université Pierre et Marie Curie - Paris6, 2000.
[13] P. Lalande. Détection du mouvement dans les séquences d'images selon une approche markovienne, application à la robotique sous marine", Thèse Université de Renne I, 1990.
[14] F. Lohier, L. Lacassagne, P. Garda, "Generic programming method for real time implementation of MRF based motion detection algorithm on a multiprocessors DSP with multidimentionnal DMA". GRETSI 1999.
[15] A. Manzanera, Vision artificielle rtinienne. Thèse ENST 2000.
[16] A. Manzanera, F. Prêteux, T. Bernard. Markovian-based modeling on programmable retina. QCAV 2001.
[17] 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, ICVGIP '04.
[18] A. Mérigot, Asociative Nets Model: a graph based parallel computing model, IEEE Transaction on Computer, 46(5), 558-571, 1997.
[19] A. Mérigot G.Constantinescu B.Ducourthial D.Dulac & S.Guezguez, Evaluation d'un modèle de calcul parallèle pour l'analyse d'images, AAA 1998.
[20] A. Mérigot & B.Zavidovique, Image analysis on massively parallel computers: an architecture point of view. nternational journal of pattern recognition and image analysis. 6(3), 2002.
[21] G. Mostafaoui, T. Kunlin, L. Lacassagne Relaxation markovienne et seuillage par hystérésis pour une détection de mouvement temps réel dans des séquences couleur, International Symposium on Image/Video Communications over fixed and mobile networks, ISIVC, Jully 2004, Brest, France.