Embedding of a Real Time Image Stabilization Algorithm on SoPC Platform, a Chip Multi-processor Approach

Jean Pierre Dérutin, Lionel Damez, and Alexis Landrault

LASMEA - UMR 6602 du CNRS,
Université Blaise Pascal,
Clermont-Ferrand, France
{derutin,damez,landrault}@lasmea.univ-bpclermont.fr

Abstract. Highly regular multi-processor architecture are suitable for inherently highly parallelizable applications such as most of the image processing domain. System on a programmable chip (SoPC) allows hardware designers to tailor every aspects of the architecture in order to match the specific application needs. These platforms are now large enough to embed an increasing number of core, allowing implementation of a multi-processor architecture with an embedded communication network.

In this paper we present the parallelization and the embedding of a real time image stabilization algorithm on SoPC platform. Our overall hardware implementation method is based upon meeting algorithm processing power requirement and communication needs with refinement of a generic parallel architecture model. Actual implementation is done by the choice and parameterization of readily available reconfigurable hardware modules and customizable commercially available IPs. We present both software and hardware implementation with performance results on a Xilinx SoPC target.

1 Introduction

In this work, we show with a practical case the embedding of a real time embedded vision application. Real time embedded vision is useful for example in mobile robotic field, where the design of intelligent sensors usually requires integration of sophisticated processing near the transducer. Complex examples would be to embed algorithms that enable human face recognition with monocular vision or a 3D localization by mono or stereo vision approaches. A simpler example described in this article is to embed a real time image stabilization algorithm[1].

In tele-operation or aided-driving systems, parasite motion suffered by a camera (bumpy ground, vibrations, etc.) may strongly interfere in the visualization process and understanding of the image sequence. In this case stabilizing an image sequence consists in eliminating or smoothing unintended motion component, while leaving the driven motion component intact.

The development of such applications imposes multiple and severe design constraints, which could be conflicting. More precisely, it is necessary to provide sufficient processing power to run the algorithms at the information flow rate, using a

J. Blanc-Talon et al. (Eds.): ACIVS 2008, LNCS 5259, pp. 157–169, 2008.

system of minimal size that consumes little power. In general, to meet these constraints, it is necessary to define a specifically dedicated electronic architecture. Today, transistor integration density enables concentration of the main part, or even all, of a complete system on a sole component (System On Chip - SoC). This type of component is composed of various components, such as standard or specialized microprocessors, memory, communication systems and various peripherals, available as 'IP' blocks.

Highly regular multi-processor architectures are known to be efficient in image processing application domain because those applications are very compute intensive and exhibit a high degree of data-level parallelism. The chosen generic architecture model (Multiple Instruction Multiple Data with distributed memory: MIMD-DM) is suitable for a homogeneous and regular network comprising communicating processor cores. We present in this article how this abstract architecture model is derived into a concrete architecture in order to meet the specific application computation and communication needs.

In section 2 and 3, we discuss several stabilization methods, then we present the stabilization application and the processing steps of the stabilization method. Section 4 describes the generic architecture used in the context of the experimental platform that we have chosen. Section 5 describes the parallel structure of the algorithm and the choices made while porting the software application into embedded software. Section 6 provides a few results concerning resource costs and the performance of the different processor and communication implementation options. And finally in section 7, we present some performance results related to the actual implementation on a Xilinx SoPC target.

2 Electronic Image Stabilization

In the last years, several electronic image stabilization methods were proposed. These methods may be classified into three main families, according to the adopted motion model: 2D or planar methods [2] and the presented method, 3D methods [3] and 2,5D methods [4]. In fact, stabilization algorithms are composed of a sequence of processing blocks, which have different levels of complexity. Generally, three main processing stages are executed:

- image matching,
- global movement estimation, using a motion model,
- compensation of the unwanted motion, getting as result a stable sequence.

2.1 Image Matching

We aim to calculate the movement in the 2D image plan of a real-world point or region. This movement is the 2D projection of the object's 3D motion in the observed scene. The most current ways to solve this problem are the optical flow extraction [3] and feature-based approaches [2]. Even if the optical flow extraction method (explained in [5] and [6]) has already been employed for image stabilization purposes, it is constraining because of its mathematical complexity, that may be relatively high. Another constraint is the assumption that the optical flow field is the 2D projection of the 3D motion [7]. In this study, to process image matching we use detection and tracking of

visual features. This method consists of two steps. First, searching in the image i for regions with strong visual information (e.g. strong luminance contrast, corners, edges, etc.) called visual features, and then identifying these same regions in image $i + 1$.

Different tools for visual features detection are known, for instance, the "corner and edge detector" [8], the Laplacian operator and Harr wavelets (the latter are presented in the next section). Once the features have been detected, we must be able to find them in another image. This task is done using a correlation method combined with a search strategy. Multi-resolution techniques allow a smaller processing time, through a "coarse-to-fine" approach. Several correlation methods may be employed, from the simplest ones being SSD (Sum of Squared Differences) and SAD (Sum of Absolute Differences) [9], to light-changes robustmethods like normalized cross correlation [10].

2.2 Global Movement Estimation

Once image matching has been achieved, we can proceed to the second stage of the stabilization processing chain, being the estimation of the motion parameters that are determined by the adopted motion model.

2D models suppose a planar or quasi-planar scene. All points tracked in the preceding stage must lie in approximately the same distance from the camera. In this case, there are three parameters to estimate: two translation movements (horizontal and vertical) and one rotation around the camera optical axis. A fourth parameter may be included, to take into account scale changes caused by the camera forward/backward motion.

3D models suppose that only rotational parasites are relevant. So, we have to estimate and correct 3D rotations to stabilize our film. Knowing that camera rotation effects in images are independent from scene depth, we are able to estimate camera rotation parameters, using quaternions for instance [2].

The 2,5 model presented in [4] presuppose the availability of preliminary information about camera motion and allows us to estimate three global motion parameters, plus one independent depth-related parameter for each analyzed (tracked) point. It allows us to work with structurally sophisticated scenes, without needing an advanced 3D model.

2.3 Motion Compensation

Finally, after estimating the global motion between images, we're going to compensate its unwanted or unintended component. This last processing stage is closely related to the application framework. The definition of "unwanted motion" depends entirely from the kind of "stability" required in each application. Several methods can be used: "full compensation" corresponding to static background scene, low-pass or inertial filtering [4] or low-order polynomial fitting for 3D rotations [3].

3 Stabilization Method

We have developed and parallelized a stabilization application, and embedded it in the presented architecture. In [1] a first sequential implementation was done and then parallelized on a cluster of workstations.

Fig. 1. Stabilization algorithm: Feature detection step (left) and pattern matching step (right)

It comprises several steps such as visual features detection using Harr's wavelets and the search of matching points between two successive images, using a SSD operator, applied to measure the similarity between the searched feature and its potential matching. Fig. 1 illustrate feature detection and feature matching steps. Once we get matching points between the two successive images, we can estimate the 2D motion model parameters (Δx, Δy and $\Delta\theta$), using the Median Least Squares Method (MLSM). Finally, the movement parameters are filtered and the obtained unwanted motion component is used to warp the respective image, stabilizing the video sequence.

From each image acquired by the camera (coded in 256 gray levels, image size adjusted by the user) three intermediary images are produced: one integral image, that will be used for visual features detection, and two sub-sampled images ($^1/_2$ and $^1/_4$ pixels), used to construct the multi-resolution searching pyramid. The integral image has in its (X, Y) position the sum of all pixels integrated inside the rectangle delimited by $i(0, 0)$ and $i(X, Y)$, when $i(x, y)$ is the original image.

Harr's wavelet processing consists of the convolution between an image region (pattern) and one of the wavelet masks. The obtained value represents the luminance gradient in a given direction. Wavelet processing is strongly accelerated when using the integral image. In this situation, we can evaluate the sum of all pixels inside a rectangle of any size performing only 4 memory access and 3 sum operations [11].

Features are detected applying the wavelets over a pre-defined zone. We use a region of the upper half of the image (size $q \boldsymbol{x} r$) to search features present in the horizon line. Normally, these regions are far away from the camera, enough to respect the planarity constraint of the 2D motion model. The detection zone is divided in $n/3$ vertical blocks (size $3q/n \boldsymbol{x} r$), n being the desired features number, set by the user (see Fig/ 1). For example, with an image size $H_{Tot} \times W_{Tot}$ = 1280×960 , $q = 1024$, $r = 384$ and the block size = 128×384 for n = 24. The three types of wavelets (vertical, horizontal and diagonal) are applied into each block, and the three regions presenting the biggest values of vertical, horizontal and diagonal gradients respectively are selected as features.

After the detection stage in image i, we seek the n corresponding features in image $i+1$. A "search window" of size $2T \times 2T$ is defined around the position where a feature

was detected. Then, SSD is calculated between each region inside this window and the detected feature in image i (see Fig. 1). The region in image $i+1$ which minimizes the SSD is considered to be the match of the respective feature. This operation is repeated for each one of the n features detected in the preceding stage, giving us n points matching between two successive images. T is the largest displacement of a feature from one given image to the next one.

To reduce the processing load, the search for matching features is executed in a multi-resolution approach. We start using a $1/4$ sub-sampled image, and looking for a SSD minimization inside a $T/2$ **x** $T/2$ window. This provides us a first estimation for the matching point position.

Based on this estimation, a second search process begins, using a $1/2$ sub-sampled image and a 3×3 search window, placed around the first estimated position. A second estimation is thus obtained, more accurate than the first one. Finally, a final search stage is executed, using the original image and a sub-pixel precision of $1/8$ pixel. A 2×2 search window is analyzed, and the luminance of regions lying between pixels is calculated through a bilinear interpolation of the adjacent pixels.

Having found the n point matching between images i and $i+1$, we are able to estimate the 2D model parameters describing the movement from one image to other. This movement can be modeled by a homogeneous transformation matrix, composed by vertical and horizontal translation movements and a rotation around the optical axis. Three matrix parameters, related to each of these movements, must be estimated. The n point matching are applied to the model, and the error is minimized using the Median Least Squares Method.

The obtained motion parameters are accumulated to those processed before, in order to find the total camera displacement since the beginning of the video sequence. The found values are filtered by first order linear filters. Each parameter has an independent filter, and all the filters coefficients may be set by the user. This method allows us to have flexible stabilization intensity, adjustable to the application. We are also able to have different stabilization levels for translation movements and rotation.

The filtered values are used to get the inverse homogeneous transformation matrix, which is applied to image $i+1$ to stabilize it, bringing it back to a dynamic reference position. This dynamic reference tries to follow the commanded camera motion, respecting the passing band determined by the filter coefficients adjusted by the user.

4 Generic Architecture Description

The chosen generic architecture model is type MIMD-DM with processing nodes communicating using message passing communication model. We choose this architecture because of its ability to execute efficiently any parallel scheme. Interconnection network is a static interconnection network, it has a hypercube-based topology structure because of its properties: for a number of nodes $n = 2^D$, diameter D and number of links/node $= \log(n)$ and total number of links is $\frac{n}{2} \times \log(n)$. This structure eases routing; which can be calculated using a simple combinatorial function, and which enables adaptive routing thanks to several potential paths between two nodes. This topology is also of great interest in terms of extensibility, as when the number of processors is doubled, the

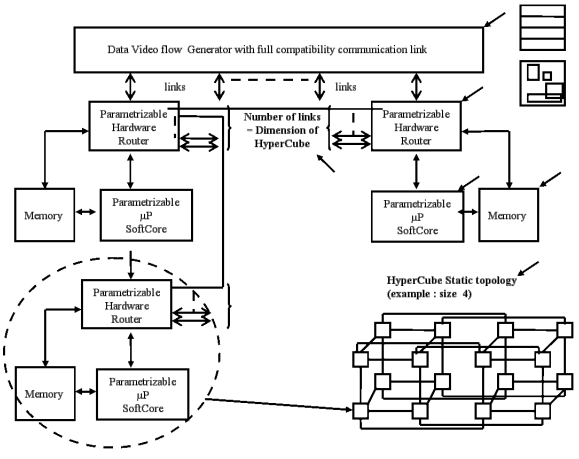

Fig. 2. Network of Communicating soft Processors featuring networked image data input

maximal distance between two processors (diameter) and the number of links per node only increases by 1.

This architecture is regular and homogeneous, each node is composed the same identical components: a processor, with local memory for application software and data storage, and some communication device. All components can be chosen inside a library of available custom components or commercial IP. Depending on the application an architecture derived from the generic architecture may differ from another through the different options and parameters relating to the processors (optional arithmetic units, implementation options), to the memory (amount of local memory, size of potential buffers) and to the interconnection network (type of link between processor, number of nodes).

Final hardware architecture can range from a very simple to complex hardware configuration. Fig. 2 shows a full feature configuration with a complex interprocessor on chip interconnection network involving a packet router and a dedicated video input communication device able to send parts of input video (Window Of Interest). In this work architectural choices were focused on Xilinx Microblaze microprocessor configurations and on a direct point to point communication link described in section 6.

5 Implementation of the Stabilization Algorithm

5.1 Parallel Implementation

In [1] the characteristics of the different application stages were analyzed in order to concentrate the parallelization efforts on those stages where the speed gain may be potentially high. The first four processing stages of the sequential version were implemented in parallel and the two last steps, not presenting an important complexity ($<2\%$), were implemented in sequential mode. The parallelized processes (feature detection and feature matching) were then implemented for SoPC target.

We can see on Fig. 3 the parallel implementation scheme. It is based on data parallelism (images then lists of points) between the N_P available processors.

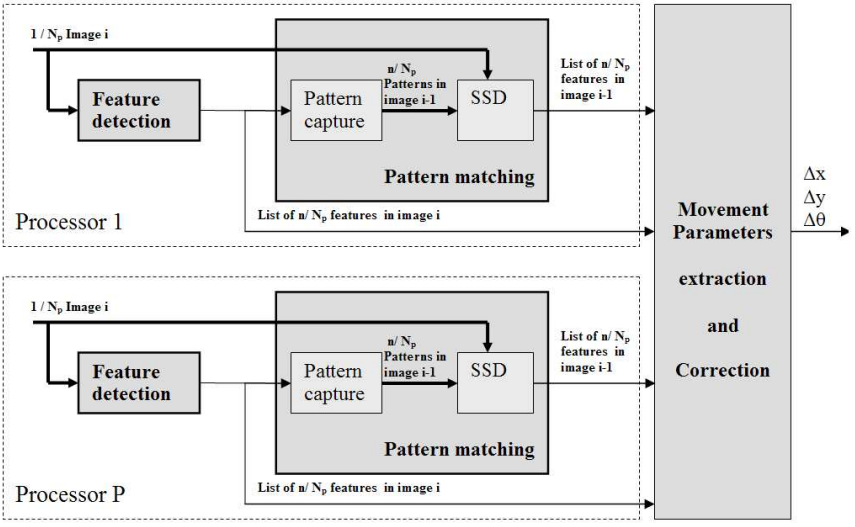

Fig. 3. Parallel implementation scheme

Detection is executed by each processor p on a block of the detection zone of image i. The block size is defined by $(q/N_P, r)$, and each processor returns a list of n/N_P primitives which will be tracked in the next stage. **Features matching** is the most time consuming stage in sequential version. In its parallel implementation, each processor does the matching search for each feature it has detected. It means that each processor is responsible for tracking its part of the desired features number (n/N_P).

5.2 Parallel Implementation on SoPC

We ported the parallel processes for our embedded architecture. An embedded hardware architecture is much more limited than a workstation in term of memory available to store the software application and the processed data. As a consequence we have made different implementation choices in the two versions.

In the first implementation of our stabilization application several techniques were employed in order to reduce processing load and/or to increase precision, some of these techniques are not used in the embedded version in order to reduce the memory requirements of the software application:

– Harr's wavelets detector is processed over a transformed image (integral image), which is calculated before the detection step. The storage of this integrated image requires a very large amount of memory: $4 \times$ memory requirement of the 255 gray level image stored in each processor as each integrated pixel must be stored using a 32 bit word. In the embedded version it is possible to calculate the wavelets detector response using directly the input image, thus avoiding the need of storing the integral image.

- The search of matching points is originally done using a multi-resolution pyramidal strategy, with three resolution levels. It is necessary to store the 1/2 and 1/4 resolution images. In the embedded version template matching is done using a direct strategy. Patern matching precision is then degraded to the pixel precision instead of a $1/8$ pixel precision.
- Finally, instead of storing both $image\ i$ and $image\ i-1$ (using a rotating buffer technique avoiding much data copy), only patterns are stored from an algorithm iteration to another, adding a *pattern capture* process inside the *pattern matching* step (see Fig. 3).

6 Hardware Implementation Results

In this section we present the effect of processor and communication network configuration in terms of SoPC resource consumption. Considering the application computational and communication needs, this will lead to a specific hardware configuration choice.

The synthesis results for each type of processor configuration or the chosen communication link are presented. Afterwards, communication performance is presented in terms of the latency and bandwidth of each solution.

6.1 Processor Configuration

Table 1 shows the implementation costs of a processor in its smallest and largest configuration. On the Virtex-4 architectures, MicroBlaze version V4.00 is used, and on Virtex-5 architectures, MicroBlaze version V5.00 is used. By comparing tables 1, it can be seen that if we choose to configure MicroBlaze in such a way that a minimum of resources is used, MicroBlaze occupies approximately 1% of the largest SoPCs. If the cost of other elements of the system is not taken into account, we could place up to 94 processors in a Virtex-4 LX200 or 118 in a Virtex-5 LX330. The use of an arithmetic unit and an FPU are the two most expensive options, as they require more than 7% of DSP block resources for a Virtex-4 LX200 and 3% of the same resources for a Virtex-5 LX330. We wanted to make the processor as small as possible, but able to execute efficient the numerous multiplications involved in the feature tracking step of the stabilization algorithm. As a consequence processor has no other optional arithmetic or logic unit than a hardware multiplier.

6.2 Point-to-Point Based Network

In order to enable communication between processors, we have also provided the choice of several communication link type, ranging from a simple FIFO point to point communication, to more complex solutions such as DMA point to point links or a Network on Chip (NoC) based on a packet router. Only the simplest one (FIFO point to point link) is presented in this paper as it satisfies the application communication needs for relatively low implementation costs. We have used the MicroBlaze Fast Simplex Link

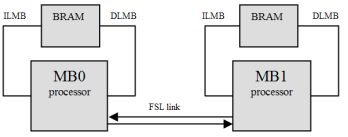

Fig. 4. FSL point to point link

(FSL), illustrated in Fig. 4, that enables data in the registers to be sent from one processor to a neighboring processor via a FIFO connection. The MicroBlaze processor has 8 output ports and 8 input ports, and it is possible to establish a full-duplex connection with up to 8 neighboring processors.

We have generated architectures based on FSL point-to-point links, by varying the number of processors. For this evaluation, we have used the smallest MicroBlaze configuration (no optional units). The following tables show the space and resources used by the complete system, using one of the FSL (table 2) point-to-point connections. FIFO depth is configured to 64 elements, and each processor has 16Kb of local memory. A solution with 32 processors, based on FSL links, easily fits a Virtex-4 LX200. We find out that this type of link clearly takes advantage of the optimization of Virtex-5 for embedding FIFOs, as the cost of the system is reduced to almost that of the processors and the memory.

6.3 Communication Performance

In order to access FSL ports, the processor reads and writes in special registers, using the operations included in its instruction set. With this type of link, data is passed directly from the registers of one processor to the registers of its neighbor. Several cases must be studied in order to obtain an accurate assessment of the level of performance of this type of communication. Performance for a MicroBlaze V5.00 is shown. If we only consider register to register communication (therefore with very limited amounts of data), this communication mode enables particularly low latency to be reached: just one clock is required.

If we change the data in the memory, the reading and writing latency must be taken into account. Hence, the sending and receiving latency of 32 bit data, situated in an internal memory block on the processor's local bus, is $2.T_{clk}$.

Table 1. Left: Available space and resources in Virtex-4 and Virtex-5 targets of higher capacity. Right: Resources occupied by the MicroBlaze processor in its smallest and largest configuration.

DEVICE	SPACE	RESOURCES			
	SLICE	FF	LUTs	DSP48	BRAM
XC4VLX200	89088	178176	178176	96	336
XC5VLX330	207360	207360	207360	192	288

Target	MicroBlaze parameters				RESOURCES			
	Shift.	Div.	Mult.	FPU	SLICE	FF	LUTs	DSP
Virtex4	0	0	0	0	936	559	1146	0
Virtex4	1	1	1	1	1777	1297	2582	7
Virtex5	0	0	0	0	1753	1058	1212	0
Virtex5	1	1	1	1	3251	1900	2400	7

Table 2. SoPC resources used for 4, 8, 16, and 32 processors with FSL point-to-point links for Virtex4 (Left) and Virtex5 (Right)

Device	Nb P	SLICE	FF	LUTs	BRAM
XC4VLX200	4	5%	1%	4%	9%
XC4VLX200	8	11%	2%	10%	19%
XC4VLX200	16	26%	4%	24%	38%
XC4VLX200	32	60%	7%	55%	76%

Device	Nb P	SLICE	FF	LUTs	BRAM
XC5VLX330	4	3%	2%	3%	6%
XC5VLX330	8	6%	5%	6%	11%
XC5VLX330	16	12%	10%	14%	22%
XC5VLX330	32	23%	17%	28%	44%

In addition, if a large amount of data is exchanged, the time taken to execute loop control instructions, which has a larger influence on the final communication time than the time actually taken to transfer, must be taken into account. For an unfolded loop it would take $0.5 \times F$ B.s^{-1}, therefore 70 MB.s^{-1} for a processor with a frequency of 140MHz. The use of software optimization techniques enables the rate of transfer to be increased slightly. We therefore have a type of link that enables very low latency communication, but for which the bandwidth is limited by processor performance.

7 Software Implementation Results on SoPC Platform

Since this application communication needs are very low and computation needs are much heavier, we have chosen the simplest hardware communication solution which uses FSL point to point links in order to fit a larger number of processors. FSL FIFO were configured to 16×32 bit elements deep and processor configuration uses a hardware multiplication which greatly increases performances of template matching (approximately a factor 3). Processor local memory is configured according to the maximum device capacity (see table 3).

To evaluate the whole flow of our approach, we have used a Virtex 4 LX 60 platform [12]. The local image window to be treated by each processor is loaded in corresponding local memory of each processor as the input/output module mechanism will not be dealt with in this paper (a hardware module for image acquisition and slicing is currently under development). Benchmark metrics are program size, image data size and application size for a 1 to 64 processors solutions. Validations for 1 to 16 processors are obtained from direct measures onto the platform. Systems with 32 and 64 processors did not fit into the test platform, performance results are obtained by co-simulation using a processor Instruction Set Simulator mixed with VHDL models. Timing results enable to evaluate application speed-up from one solution to another (depending on the number of processors implemented) and run-time of the application.

7.1 System Dimensioning

We can see in table 3 (right) the size of the local image window($H_{PE} \times W_{PE}$) and the number of primitives processed by each processor. Table 3 (left) shows the memory requirements of the application depending of the number of processors in the system. The

Table 3. Left: Application memory requirements versus available local memory per processor for 1 to 64 processors in KB. Right: Application parameters variation with number of processors.

Number of processors	1	4	8	16	32	64
program size	3.76	4.54	4.71	5.30	5.64	5.74
image data size	99	27	15	9	6	4.5
Total Application Size	**102.76**	**31.54**	**19.71**	**14.30**	**11.64**	**10.24**
available memory (KB) XV4LX60	256	64	32	16	8	
available memory (KB) XV4FX140 & XV5LX330	1024	256	128	64	32	16

Number of processors	1	4	8	16	32	64
H_{PE} × W_{PE}	528 × 192	144 × 192	80 × 192	48 × 192	32 × 192	24 × 192
Image division factor	1	3,66	6,6	11	16,5	22
nb of features per processor	64	16	8	4	2	1

amount of available memory for the software decreases with the number of processors, which can lead to a too large code size for a given hardware configuration. Application memory requirements are due to program code and image data. Program code, instead of reducing with the number of processors, is increasing as communication code requirements becomes larger with the number of processors. With the chosen data partitioning of the parallel application, image data size decreases with number of processor by a factor shown in table 3. We can see in table 3 that system up to 64 processors with our stabilization application can be embedded in the largest SOPC target such as a Virtex4FX140 or virtex5 LX330 as they have sufficient embedded memory and up to a 16 processors configuration can fit our test platform which features a Xilinx Virtex4LX60.

7.2 Timing Performances

The timing performances of the stabilization were measured by processing an input image in the size $H_{Tot} \times W_{Tot} = 528 \times 384$ and by configuring the algorithm to search 64 visual primitives at a distance T of 6 pixels. We have tested system up to 16 processors in a Xilinx XV4LX60 device. Results for 32 and 64 processors are simulation results. A Microblaze Instruction Set Simulator developed in DEPECA[13] has been used in order to set up a co-simulation environment and speed-up the simulation process. We can see in left table 4 that with 16 processors and more the parallel phases of the application execute in much less than 40ms, leaving time for execution of the whole algorithm (with the sequential parts) to process 25 images/s. For 32 processors the processing times are

Table 4. Application execution time (ms) and application speed up for 1 to 64 processors

Number of processors	1	4	8	16	32	64
Detection time	217.152	55.919	27.965	13.968	6.928	3.468
Tracking time	31.787	7.948	3.975	1.988	0.992	0.497
Communications time		0.016	0.016	0.016	0.016	0.016
total time	**248.939**	**63.883**	**31.956**	**15.972**	**7.936**	**3.981**

Number of Processors	1	4	8	16	32	64
Detection Speed up	1.00	3.88	7.78	15.55	31.34	62.72
Tracking Speed up	1.00	4.00	8.00	15.99	32.04	63.00
Total Speed Up	**1.00**	**3.90**	**7.79**	**15.60**	**31.36**	**62.53**

small enough (8ms) to consider using this algorithm as a preprocessing for higher level vision algorithms.

Speed up shown in right table 4 are very near to linear for both detection template matching step. For 32 processors, speed-up are even slightly surlinear for template matching, which seems due to the approximations from the co-simulation processor model.

8 Conclusion

This article presents the implementation of a realistic image processing application on a SoPC target. It shows that an homogeneous network of processor is efficient for embedded image processing. Processing times and application speed-up are very interesting. A very near to linear speed-up and a scalable architecture make possible to match the processing power with the input video data rate by adjusting the number of processor. Future works include extension of range of choice regarding processing elements. For example, we are working on the possibility to integrate and parameterize OpenFire[14] soft processor. Future works also include implementation of more complex image processing applications. The chosen applications should require more communication power. It will allow us to show another type of communication device (a on chip packet-router) which is not presented in this paper. The ultimate goal being to make the processor and communication system first choice elements for tuning the architecture to a given application needs.

References

1. Derutin, J.P., et al.: Simd, smp and mimd-dm parallel approaches for real-time 2d image stabilization. In: CAMP 2005. Computer Architecture for Machine Perception, pp. 73–80. IEEE Computer Society, Los Alamitos (2005)
2. Morimoto, C.: Electronic Digital Stabilization: Design and Evaluation, with Applications. Phd thesis, University of Maryland (1997)
3. Duric, Z., et al.: Shooting a smooth video with a shaky camera. Machine Vision and Applications 13, 303–313 (2003)
4. Zhu, Z., et al.: Camera stabilisation based on 2.5d motion estimation and inertial motion filtering. In: International Conference on Intelligent Vehicles (1998)
5. Horn, B., et al.: Determining optical flow. Artificial Intelligence 17, 185–204 (1981)
6. Barron, J., et al.: Performance of optical flow techniques. International Journal of Computer Vision 12, 43–77 (1994)
7. Verri, A., et al.: Motion field and optical flow: Qualitative properties. IEEE Trans. Pattern Analysis and Machine Intelligence 11(8), 490–498 (1989)
8. Harris, C., et al.: A combined corner and edge detector. In: Proceeding of the 4th Alvey Vision Conference, pp. 147–151 (1988)
9. Pourreza, H., et al.: Weighted multiple bit-plane matching, a simple and efficient matching criterion for electronic digital image stabilizer application. In: 6th International Conference on Signal Processing, vol. 2, pp. 957–960 (2002)
10. Tsai, D., et al.: The evaluation of normalized cross correlations for defect detection. Pattern Recognition Letters 24, 2525–2535 (2003)

11. Viola, P., et al.: Rapid object detection using a boosted cascade of simple features. In: Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (2001)
12. AVNET: Xilinx virtex-4 lx evaluation kit (ADS-XLX-V4LX-EVL60-G) (2008), http://www.em.avnet.com
13. Mateos, R., et al.: Hardware/software co-simulation environment for csoc with soft processors. In: IEEE Internacional Conference on Field-Programmable Technology ICFPT 2004, pp. 109–114 (2004)
14. Craven, S., et al.: Configurable soft processor arrays using the openfire processor. In: Proceedings of 2005 MAPLD International Conference, pp. 250–256 (2005)