Efficient Implementation of Pattern Matching Recognition in Heterogeneus Architectures

Javier Gonzalez-Bayon, Pablo Sanchez, Javier Barreda
University of Cantabria, TEISA department
Avda de los Castros, s/n, Santander
Spain
{javiergb, sanchez, barreda}@teisa.unican.es

***Abstract*—Implementation of pattern recognition algorithms is an important and complex task. These schemes usually perform time and resource consuming operations as 2D FFT. That is the reason why usually they are implemented in heterogeneous platforms with at least one DSP that accelerates the mathematical applications. Nevertheless, even using this specific processor the execution time required can be excessive. Therefore, an efficient use of all the processors available in the platform is mandatory. This paper proposes an efficient implementation of pattern recognition schemes in a heterogeneous platform that includes an ARM processor and a DSP.**

Keywords—embedded implementation; heterogeneous architecture; pattern matching recognition

I. INTRODUCTION

The recognition of an object independent of its position, size and orientation is an important task in pattern recognition. In the last two decades a number of techniques have been developed to extract image features which are invariant under translation, scale change and rotation caused by the image formation process. A good recognition scheme is desired to be discriminative, robust, and computationally inexpensive in both terms of time and storage requirement.

It is usual that these recognition algorithms have to be implemented in handheld devices with battery limitations. Thus, it is important that their implementation avoids excessive computations or power consumption. And it is usually complex to translate the mathematical functions of these algorithms to the target architecture. The complexity arises for heterogeneous platforms where more than one processor is available. This paper focuses in an architecture that is composed of an ARM processor and a DSP. To obtain the maximum performance for this platform is necessary to use adequately the resources available. For example, boards that include DSP have a specific library for performing specific mathematical operations. Thus, it is necessary to comprehend it.

The pattern recognition application implemented in this work consists in the recognition of the position of several clocks of a picture. An image example can be observed in Fig. 1. There are six different clocks to measure. For a visual recognition application like this one, usually the FFT (Fast Fourier Transform) is the more important and used function.

This work has been supported by Project TEC2011-28666-C04-02 given by the Spanish ministry of Economy and Competitivity.

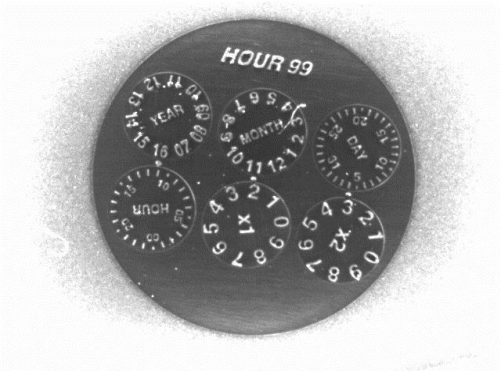

Figure 1: Picture for the recognition application

How to perform this function is one of the keys to obtain a rapid application implementation.

The paper is organized as follows. In Section II, the state of the art is analyzed. The target architecture is presented in Section III while the algorithm is shown in Section IV. The implementation proposal is described in Section V. Some results are shown in Section VI. Finally, the conclusions are drawn in Section VII.

II. STATE OF THE ART

In [1] an automatic human background substitution system algorithm is implemented on a multi-core processing architecture. Compared to a single processor implementation, the experimental results show significant speedup ratio of the parallelized system on a multi-core embedded platform, which consists of an ARM processor and two DSP cores.

The article in [2] presents an efficient implementation of the algorithm on a multi-processor architecture. The algorithm-architecture aims to optimize the implementation of the algorithm on a low-cost and heterogeneous architecture (an ARM processor with SIMD coprocessor and a DSP core).

In the paper in [3], they describe synchronization counters, a mechanism that allows seamless implementation of low-latency multiprocessor synchronization. This mechanism is used in heterogeneous multiprocessor environments even when the individual processing elements lack native synchronization support. The SoC contains three special

purpose DSPs and an ARM application processor, sharing system memory and peripherals.

Design and implementation of embedded parallel DSP software poses various challenges due to the complexities posed by the application as well as heterogeneous nature of the platform. One of most issues with regards to such system is communication between processors. In [4], they present a reusable message passing software framework to make application level code more portable. The software framework is applied to an ADSP-TS101 DSP multi-processors platform.

III. TARGET PLATFORM

A BeagleBoard XM [5] is chosen as the target platform in this work. This platform includes a SoC of the OMAP family from Texas Instrument with two cores: an ARM Cortex A8 and a DSP 65+ with a maximum clock rate of 1GHz.

The API Codec Engine available by Texas has to be used to execute code in the DSP. To perform the call to the code developed, an API layer called VISA is used. The structure of the software stack is defined using the Codec Engine that is shown in Fig. 2. Inside the layer VISA there are different types of interfaces, mainly oriented to processing audio and video. In this paper, the IUNIVERSAL interface will be used since it can work with any kind of data.

It is important to pay special attention to how to use the DMA (Direct Memory Access) interface. It is necessary to program the DMAs to allocate adequately the data that is processed in the DSP. This data has to be stored in the cache of the board. It is also needed to store intermediate results from the algorithm implementation in this cache. This programming is vital to obtain a fine DSP performance because high data traffic through the DMAs will increase the total execution time of the algorithm.

In addition, it is needed to reserve a special memory device called CMEM that allows the reservation of memory in contiguous positions. By using this tool, the DMA operations performed with the DSP will be more efficient. Furthermore, it is necessary to describe a memory map that includes the part of the algorithm that is executed in the DSP, the buffers to store the intermediate data, the communication system and the operative system.

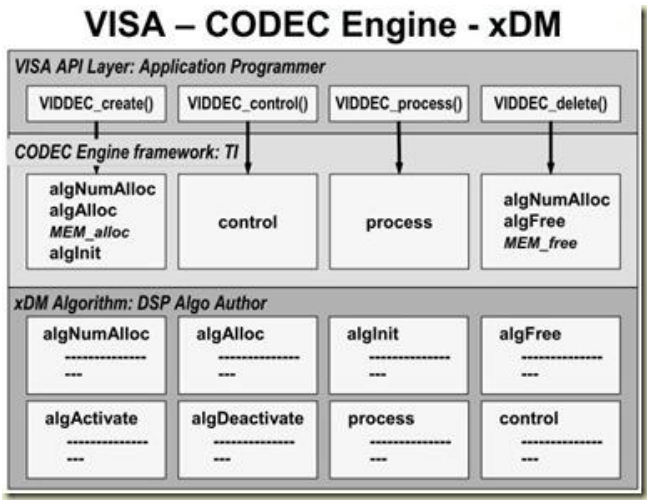

Fig 2: API layer used with the BeagleBoard

IV. ALGORITHM

There is a first version of the application code written in C and tested in a PC. Unfortunately, although it was obtained a rapid application when executing in the PC (with a total execution time of less than 1 second), the final architecture is a handheld device with the board explained in Section III. The execution time of this code in the target application increased eight times. Thus, the original code has to be modified to improve its performance in the board. In order to accomplish that, an analysis of this code is needed.

The core of the recognition algorithm is composed of two main stages. The first part of the algorithm calculates the angle necessary to rotate the whole picture (Fig. 1), so all the clocks are allocated in horizontal position. To estimate that, the image is rotated in 23 steps (accomplishing a rotation of 180º) and a search of a pattern with the text "HOUR 99" is performed. It is possible to observe this text in the image of Fig.1 and it is used to establish a first reference of where are located the six clocks.

Once the algorithm has found the approximated angle of the image compared to the horizontal position, the process is repeated (4 more rotations) but just around the first estimated angle. Thus, a finer estimation is obtained. After establishing

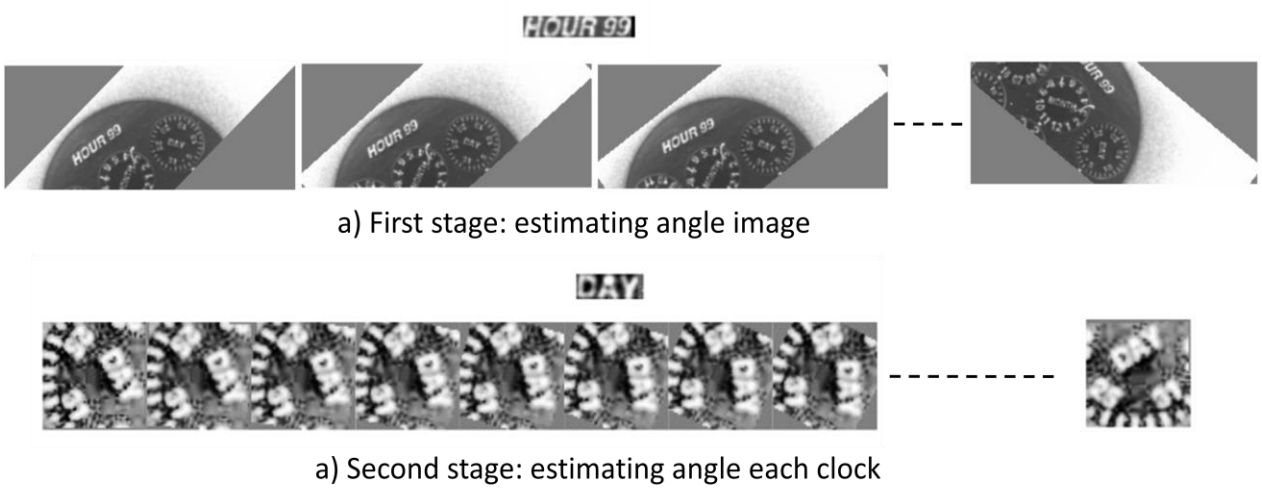

Fig. 3: The two stages of the algorithm recognition

the final angle, it is possible to locate the position of the 6 clocks with simple trigonometric functions.

The second stage is to recognize the central part of each clock. The objective now is to determine the rotation for every clock (this rotation is different for each of them). So, the algorithm performs 90 rotations per clock (with an angle of 4º per rotation) and compares the reference pattern (that is the text inside of each clock: YEAR, HOUR…) with the image of each clock rotated.

Once all the rotations for each clock are performed, it is possible to determine which is the angle of each clock and to calculate which is the value that each clock is pointing. It is important to remind that this process is performed for each of the six clocks. In Fig. 3 is shown what are the patterns and images involved in this application. Therefore, the algorithm performs 90x6=540 searches for the clocks, plus the 27 (23+4) initials searches to find the pattern "HOUR 99". It is important to explain that each of these searches is composed of two steps: a rotation and a matching estimation.

The rotation is performed by using bilinear interpolation. The final goal is to rotate around the center of the image. For a rotation over an angle φ, a point (x, y) in the original image is mapped onto the point (x', y') in the resultant image. The relation between the points is:

$$x' = x\cos(\varphi) - y\sin(\varphi)$$

$$y' = x\sin(\varphi) + y\cos(\varphi)$$

In a geometric transform we need to express x' and y' as functions of x and y. It is necessary to rotate over $-\varphi$ to get from (x', y') to (x, y). Thus, it is obtained:

$$x = x'\cos(\varphi) + y'\sin(\varphi)$$

$$y = -x'\sin(\varphi) + y'\cos(\varphi)$$

The rotation algorithm then enumerates all points (x', y') in the resulting image, calculates the corresponding point (x, y) in the original image and uses a bilinear interpolation method to estimate the value of the original image in (x, y) and assigns that value to the point (x', y').

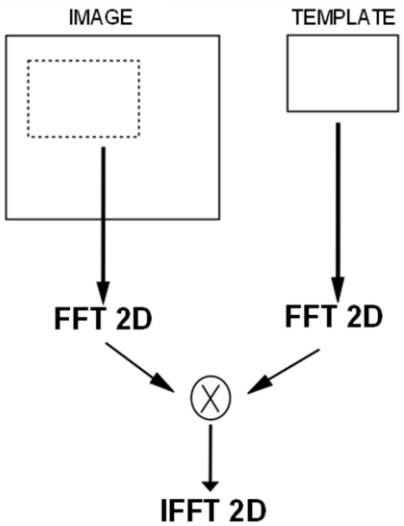

Figure 4: Matching (correlation)

The matching consists in searching similitudes between a pattern (a stored and fixed image) and the rotated image. This matching is accomplished by performing a correlation (in the frequency domain, since it is less consuming to implement than in the time domain) that consist in performing a 2-dimension FFT to both the rotated image and the pattern, to perform a complex multiplication to both outputs of the 2D FFTs and, finally, to perform and inverse 2D FFT. This correlation process can be observed in Fig. 4. This matching has to be performed several times until the whole picture (for each rotation) has been compared to the template. The final matching is obtained for the zone of the image that obtains the highest value in the correlation. This is the pseudo code that implements this matching-correlation process:

A=FFT2D image

B=FFT2D template

C=Complex Conjugate Multiplication (A*B')

IFFT(C)

In the original code, several complex 2D FFTs, among other functions, were performed. These mathematical functions were performed by using the *openCV* library. This library is oriented to programming functions aimed at real-time computer vision. The time that the algorithm spent in the 540 + 27 = 567 searches is the 75% of the total application executed in the PC. Therefore, the rotation plus matching performed for each search is the most consuming part of the algorithm. The main aim of this work is to reduce this time by implementing the algorithm in the BeagleBoard platform already mentioned.

V. IMPLEMENTATION PROPOSAL

Since the algorithm is going to be implemented in a board that includes an ARM processor and a DSP, it is important to take advantage of these resources. Therefore, two important adjustments to the C version of the code are proposed in this paper: algorithmic optimizations and parallelization. They are explained in the rest of this section.

A. Algorithmic optimizations

One of the problems of the first code version was that while the *openCV* functions is being executed in a low amount of time for *Intel* processors, the time execution increases dramatically for the ARM processor of the target architecture. The DSP can not even be used with this first version of the code, since it needs special functions. Thus, it is necessary to create an ad-hoc code to obtain a fine final implementation.

Another reason to create this new code is because the DSP, even admitting "*float*" and "*int*" data types, it does not have specific instructions for the "*float*" type and has to emulate them, needing more time to accomplish that. The original code uses "*float*" type to avoid loss of precision so a direct implementation of this code in the DSP would increases the execution time dramatically.

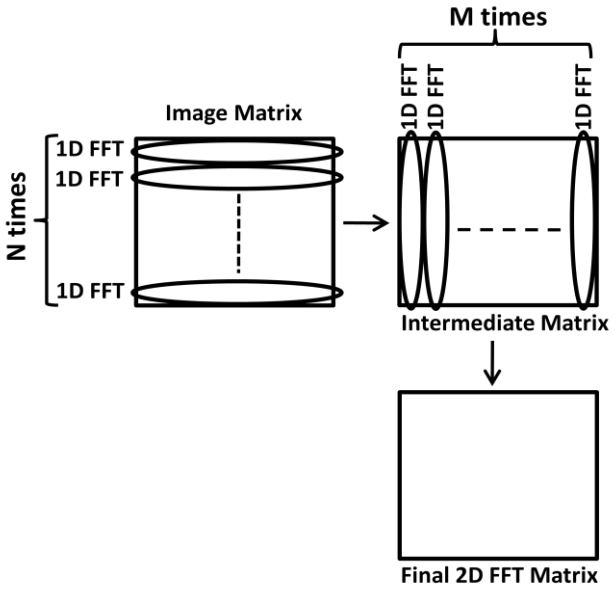

Figure 5: 2D FFT with 1D FFT functions

In addition, the code implemented in the board will have to use the FFT defined in the library of the DSP (*DSP_fft32x32_cn*, Texas Instruments). This FFT can be radix-2 or radix-4 so it works with data sizes power of 2. Also, in the library from Texas Instruments there are available FFT versions of 32 or 16 bits. In this work it was chosen data size of 32 bits because there was a huge loss of precision with data size of 16 bits. For that size, most of the clocks estimations were wrong.

The FFT available in the DSP library is of one dimension so the 2D FFTs needed for the algorithm has to be built using this one. The Fig. 5 shows how to build a 2D FFT by using 1D FFT function. If an image has size of *N*x*M*, then to perform one 2D FFT of that image requires to perform (*N*+*M*) FFTs of 1 dimension.

Taking all this into account, different optimizations have been applied to the final implementation with respect to the *OpenCV* version in this work. They are explained in the next points:

a) The FFT of the pattern is not performed every time a new matching process is performed. This is possible since this pattern is fixed and does not change as opposite to the window- image (Fig. 4) that is changing with the search. So, instead of performing 90*6=560 2D FFTs of the template, it is only performed once per clock, thus it can be reduced from 560 to 6. This is shown in Fig. 6.

b) Rewriting of the code when implementing the inverse 2D FFT. The algorithm performs the matching (explained at high level in Fig. 4) using 1D FFTs as shown in the Figure 7. But it is necessary to store the intermediate values of the matrices and the use of the DMAs increase the total execution time. So, it is possible to join the calculation of the column 1D FFT with the column of 1D inverse FFT as it is shown in Figure 8. This allows the reduction of the writings and therefore the use of the DMAs of the target platform.

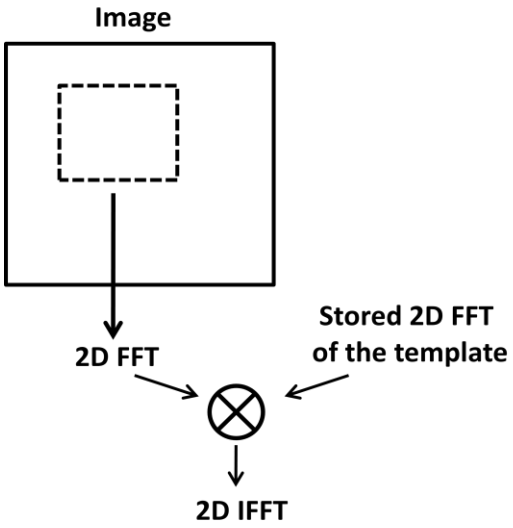

Fig. 6: Storing the FFT of the template

c) Reduction of the number of 1FFTs performed in the last step of the 2D FFT process. This is shown in Fig. 9. This allows the reduction of the number of iterations compared to a general case. This is possible because thanks to the optimization *b*) applied, part of the output of this last 1D FFT can be discarded since it contains no useful data. This reduction depends of the size of the image.

d) Algorithms to perform forward and inverse Fast Fourier Transforms (FFT and IFFT) typically assume complex input and output data. However, many applications use only input real-valued data as it is this case. But it is possible to use the complex-valued FFT and IFFT algorithms to efficiently process real-valued sequences length N/2 FFT and IFFT computation instead of length N.

Since the input of the first 1D FFT in Fig. 5 is real (it does not have imaginary part), it is possible to reduce the size of the FFTs when performing FFTs of 1 dimension as it was

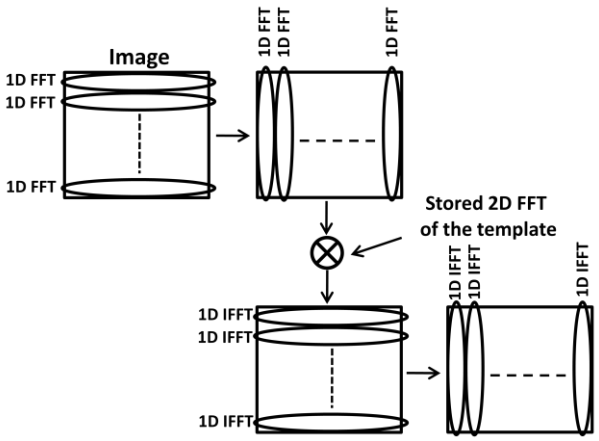

Fig 7: Matching process

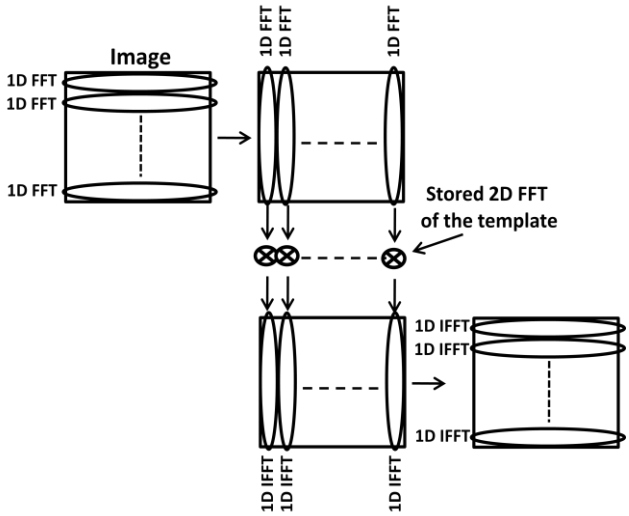

Fig. 8: Optimization *b*)

explained in [6].

e) When escalation is needed, instead of using the operator division "/" it is proposed to use the operator shift (>>). This solution uses fewer resources of the target board when performing division by 2.

f) Search of 90º instead of 360º for the 6 clocks. This optimization takes into account the trigonometric properties to avoid performing the rotation from 0 to 360º. So, only rotations from 0 to 90º are calculated and the rest correlation outputs are estimated thanks to the trigonometry properties nd it is avoided the use of the rotation function.

B. Parallelization

Parallelization has to be performed taking into account both the ARM and the DSP. Since the search is the most consuming process and it is divided in 2 steps (rotation and matching-correlation) it would be ideal to perform these steps in parallel. The ARM will perform the rotation and the DSP the correlation

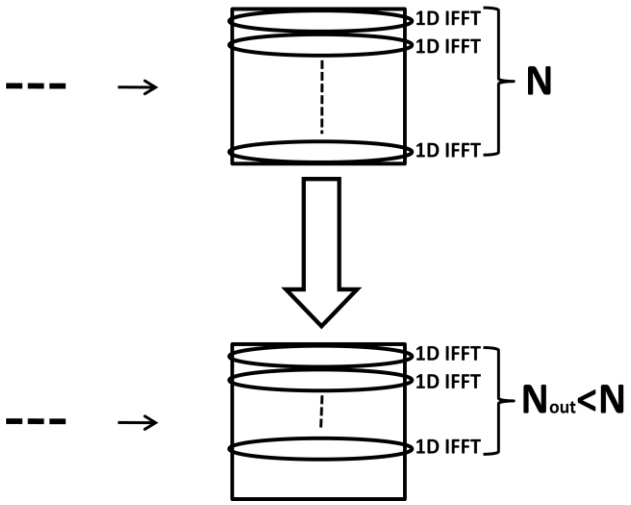

Fig 9: Optimization *c*)

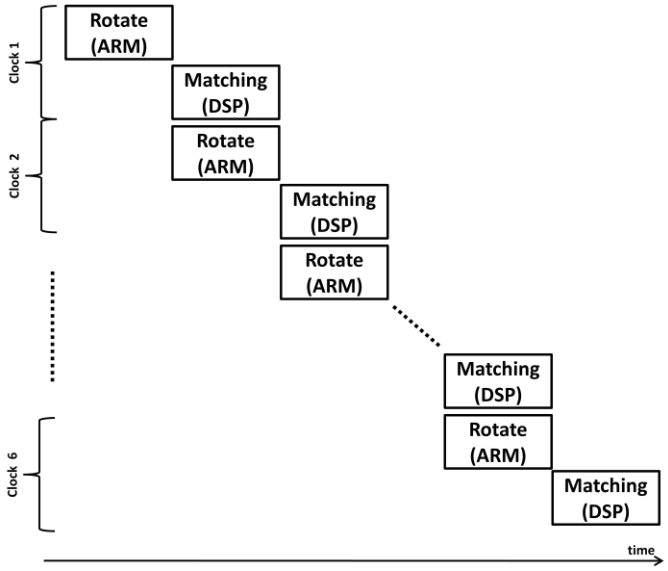

Fig. 10: Proposed parallelization

since it has specific functions to perform this step more rapidly. This is shown in Fig. 10.

The code for the correlation performed in the DSP is already optimized from the algorithmic optimizations explained in last subsection. By reducing the number of FFTs and the access to DMAs a rapid correlation is obtained. Moreover, when first measurements were performed at the final platform with the parallelization observed in Fig. 10, it was observed that the rotate performed in the ARM needed more time than the correlation performed in the DSP. Therefore, it was necessary to accelerate the rotation performed in the ARM.

This effect was not observed in the original version, since this is a problem associated to the target architecture that has fewer caches memory than in the PC. To improve the performance, instead of the classical rotation, the rotation by deformation or *shears* has been used. It is explained now.

A shear in the X direction produces y':= y (unchanged) and x':=x+α y. It turns out that any orthogonal transformation can be performed by a combination of a shear along one axis, then a shear along the other axis, followed by another shear along the first axis.

Doing a rotation by performing three shear operations might be advantageous, because it is easy to do a shear operation. To do a shear operation on a raster image (that is to say, a bitmap), there is just to shift all the pixels in a given row (column) by an easy-to-calculate displacement. But in order to do a rotation using shears, it is necessary to calculate the values of α, β, and γ from the rotation angle Θ. The original matrix rotation s is:

$$Rotation = \begin{bmatrix} \cos(\Theta) & -\text{sen}(\Theta) \\ \text{sen}(\Theta) & \cos(\Theta) \end{bmatrix}$$

Set the rotation matrix equal to the product of the three shears:

$$\beta = \sin(\Theta)$$
$$1 + \alpha\beta = \cos(\Theta)$$
$$1 + \alpha\,\text{sen}(\Theta) = \cos(\Theta)$$
$$\alpha = \frac{(\cos(\Theta) - 1)}{\text{sen}(\Theta)}$$
$$\alpha = -\tan(\Theta/2)$$
$$1 + \alpha\beta = 1 + \beta\gamma$$
$$\gamma = \alpha$$

Therefore, the angle that has to be introduced in the deformations is:

$$\alpha = -\tan(\Theta/2)$$
$$\beta = sen\ (\Theta)$$
$$\gamma = -\tan(\Theta/2)$$

This rotation has a loss of precision in the data while the angle gets higher. That is the reason to limit its use to angles between -45 and 45 degrees. To achieve the full rotation, a prior rotation of 90, 180 or 270 is performed if needed. For these angles, the rotation only implies the change of the position of the pixels (transposition of rows and/or columns).

VI. RESULTS

In this section the performance obtained for the application in the Beagle board is shown. First, the performance using the FFT from the library of the DSP and the performance of the FFT (using an equivalent function to the one used in the DSP library) implemented in the ARM processor of the board are shown in Fig. 11. It can be observed that the number of cycles needed for the FFT in the DSP increases almost linearly while by using the ARM this increase is exponentially. Thus, it was clear that the FFT (that is the core of the matching process) should be implemented in the DSP of the board.

The final execution time results obtained by different versions of the complete algorithm are shown in Table I. These versions are the initial version (PC oriented), the version with the optimizations described and, finally, the version including the optimizations and also the parallelization. It is possible to observe that the initial version is the one that obtains the worst time. That is because it uses library functions from *openCV* and also the "*float*" data types. And, as it was already explained, they are not optimal for the target platform of the project. But by creating a code with the optimizations explained in this paper, it was possible to reduce almost four times the execution time. This is because of the reductions of the most expensive functions and also to the reduction of the memory transactions that produces special time penalty in heterogeneous platforms. Finally, by implementing the algorithm in parallel (specifically, rotation in the ARM and matching in the DSP) it was possible to reduce the execution time to an additional 62%. To obtain this reduction it was necessary to choose a different rotation algorithm than the initial one.

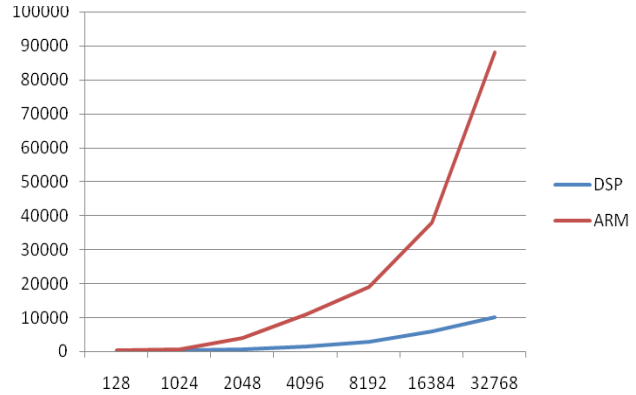

Fig. 11: Comparative of FFTs performance

TABLE I. IMPLEMENTATION RESULTS

	Original code	***Optimizations***	***Optimizations + parallelization***
Execution time (seg)	8	2.1	1.3

VII. CONCLUSIONS

In this work, a pattern matching recognition algorithm is implemented in a heterogeneous platform that includes a DSP and an ARM processor. A modification of the first version of the code (PC oriented) is needed to obtain a fine implementation for the target platform. Two kind of main modifications are proposed in this paper to accomplish this. First, some algorithmic optimizations are proposed to reduce the number of the most time consuming operations (FFTs) of the algorithm and also to reduce the number of memory transactions that would increase the total execution time. And second, the parallelization of the code. In this case, the rotation is performed in the ARM while the matching is performed in parallel in the DSP. Performance results show the adequacy of the proposed implementation.

REFERENCES

[1] Lee, Y., Chiang, C.-K., Su, T.-F., Sun, Y.-W., Kuan, C.-B., Lai, S.-H. , "Parallelized background substitution system on a multi-core embedded platform," Proceedings of the International Conference on Parallel Processing Workshops , 2012 , pp. 530-537.

[2] Vincke, B., Elouardi, A., Lambert, A., "Real time simultaneous localization and mapping: Towards low-cost multiprocessor embedded systems" Eurasip Journal on Embedded Systems, art. no. 5, 2012.

[3] Moudgill, M., Kalashnikov, V., Senthilvelan, M., Srikantiah, U., Li, T.-P., Balzola, P., Glossner, J., "Synchronization on heterogeneous multiprocessor systems," Proceedings - 2009 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, IC-SAMOS 2009 , art. no. 5289224 , pp. 133-139.

[4] Wang, X.-M., Xu, R.-Z., Qiu, F., "A software framework of message passing for parallel computing on embedded DSP systems," Proceedings - 2009 International Conference on Computational Intelligence and Software Engineering, CiSE 2009 , art. no. 5365951.

[5] http://www.ti.com/

[6] http://processors.wiki.ti.com/index.php/Efficient_FFT_Computation_of_Real_Input