Hardware/Software Co-design for Image Cross-Correlation

Mauricio A Dias* and Fernando S Osorio

University of Sao Paulo - USP
Institute of Mathematics and Computer Science - ICMC
Mobile Robotics Lab - LRM
Sao Carlos, Sao Paulo, Brazil
{macdias,fosorio}@icmc.usp.br
http://www.lrm.icmc.usp.br

Abstract. Cross-correlation is an important image processing algorithm for template matching widely used on computer vision based systems. This work follows a profile-based hardware/software co-design method to develop an architecture for normalized cross-correlation coefficient calculus using Nios II soft-processor. Results present comparisons between general purpose processor implementation and different customized soft-processor implementation considering execution time and the influence of image and sub-image size. Nios II soft-processors configured with floating-point hardware acceleration achieved a 8.31 speedup.

Keywords: Cross-Correlation, Hardware/Software Co-Design, Profile-Based Method

1 Introduction

Cross-correlation is a widely used algorithm for image processing, computer vision [10] [25] [24] and visual robotic navigation fields [20]. The main goal of this algorithm is to find the portion of an image that is most similar to a sub-image. This process can be extremely time-consuming depending on the size of the image and sub-image because the method calculates the coefficient for each pixel of the image using a sliding window that sweeps across the whole image searching for a match. Coefficient values indicate the degree of similarity between the sub-image and the current portion of an image.

Normalized cross-correlation (NCC) is a modification of the original algorithm that normalizes the coefficient values between -1 and 1. In this case, coefficient normalization makes the algorithm invariable to image and sub-image changes on brightness and scale. Visual navigation for mobile robotics is an

* The authors acknowledge the support granted by CNPq and FAPESP to the INCT-SEC (National Institute of Science and Technology of Critical Embedded Systems Brazil), processes 573963/2008-8 and 08/57870-9. Also acknowledge CNPq for financial support of this research.

example of current researching area that uses cross-correlation algorithms for visual-based navigation methods[15]. Researchers[24] suggested that the speedup of execution can be obtained using a specific hardware for this purpose.

In these work we have particular interest in the usage of the NCC algorithm applied to visual navigation for mobile robotic. Some facts should be considered in this type of application, once mobile robots are embedded systems which have some particularities and constraints. In most cases embedded systems have limited capacity of processing and memory. Embedded systems development for robotics is a complex task because these systems share computational resources with sensors data acquisition and processing, decision and actuators control. The combination of these facts justifies the adoption of more robust techniques considering performance, costs, energy consumption, processing and execution time.

The first and most intuitive solution for embedded systems design is to implement algorithms directly in hardware. Entire hardware designs solutions nowadays have been replaced by hardware/software co-designs [16]. Hardware/software co-design is a design method that proposes a hardware and software concurrent development. This method is used to decide which part of the system will be implemented in hardware and which will be implemented in software. This classic problem is called hardware/software partitioning.

Choice for adopting hardware/software co-design is based on increasing complexity of embedded systems and reduction of time-to-market for embedded systems design. Also the increasing availability of complex hardware with lower costs is creating a phenomenon known as the SoC (System on a Chip) design gap [1]. Complexity of available hardware and embedded systems projects is the cause of this gap that can be softened by new approaches as hardware/software co-design [1]. Co-design methods usually require flexible development tools that allow rapid prototyping. One way to achieve desired flexibility level is to develop hardware with reconfigurable computing[3] using devices such as FPGA's[3].

FPGA's are flexible devices because, among various features, allow hardware to be described using Hardware Description Languages (HDL's) and several hardware reconfigurations (custom hardware design). In this work specifically, there is a free soft-processor [31] provided by FPGA's manufacturer that can be used for co-design development.

A soft-processor is an Intellectual Property (IP) core which is 100% implemented using the logic primitives of the FPGA. Other definition is: a programmable instruction processor implemented in the reconfigurable logic of the FPGA [31]. Soft-processors have several advantages and one of the most relevant for actual designs is the possibility to implement multi-core solutions defining the exact number of soft-processors required by the application. Since it is implemented in configurable logic (FPGA), a soft-processor can be tuned by customizing its implementation and complexity, by changing its internal configuration, processor functional blocks or even adding new instructions to the processor, in order to match the exact requirements of an application [31].

The remainder of the paper is organized as follows: section 2 presents some recent hardware development using NCC. In section 3, concepts of hardware/software co-design and profiling-based methods are presented. Section 4 contains proposed method description followed by the results section (5). At last section 6 presents authors' conclusions based on achieved results, followed by selected references.

2 Related Works

This work is related to hardware/software co-design, profile-based methods, image processing, reconfigurable computing and vision navigation for mobile robotics areas. In these areas there are important recent works and some of them are presented in this section.

Birla & Vikram [2] presents an architecture that uses partial reconfiguration for computer vision algorithms. Hardware is developed for people detection and the reconfiguration is used to choose between using the whole image or select some specific features extracted from the image. In [17] Kalomiros & Lygouras developed a reconfigurable architecture used to select important features for robot navigation. These features are points selected on images by an image consistency verification system. Visual vocabularies were used online in [22] to detect already visited regions. These vocabularies associate images that have similar characteristics.

Lee et. Al. [18] analyzes the future of hardware and software technologies related to visual computing. Main areas of their analysis are video coding, video processing, computer vision and computer graphics. Authors' conclusions indicate that algorithms and architectures optimization should consider concurrent operations, demonstrating the importance of parallel processing in this type of application. Lu et. Al. [19] also presents the use of GPU solutions for real-time image-based rendering. This kind of hardware is usually placed on general purpose computers but can also be used when a huge amount of processing is needed.

Other works present different cross-correlation coefficient calculus [8][28][5]. These works have poor comparisons between the proposed method and the common NCC coefficient. The proposed algorithm, FNCC (Fast normalized cross-correlation), is an implementation that uses a table of values that consumes additional significant amount of memory, usually not available on embedded systems.

Recent works, [32] and [27], present dedicated hardware architecture solutions (not programmable) for NCC algorithm, that achieved acceptable execution time. In this case some parts of proposed architectures are not detailed. Recent results of other applications for NCC algorithm can be found in [9][12][23]. Next sections presents some important concepts related to this work's development.

3 Hardware/Software Co-Design

Electronic systems nowadays have become more complex because of the rising number of functionalities that are requested on projects. As electronic circuits become more complex, technical challenges in co-design will also increase [29]. These circuit systems can be classified as a joint operation component group developed for some task resolution [13] and, this components, can be implemented in hardware or in software depending on what are the system's constraints.

Hardware/Software co-design means the union of all system goals exploring interaction between hardware and software components during development [26]. This method tries to increase the predictability of embedded system design by providing analysis methods that tell designers if a system meets its performance, power, size and synthesis methods that let researchers and designers rapidly evaluate many potential design methodologies [29].

Hardware/Software co-design is a hard task and computational tools appeared to make this work faster, and also increase the system's degree of optimization. The introduction of programmable devices like FPGA's (Field Programmable Gate Arrays) on circuit co-design enabled flexibility and made prototyping easier. This fact is really important because the circuit can be developed and tested before be manufactured, reducing costs and design time. This flexibility opens new digital circuit applications and the rise of new Hardware/Software co-design problems [21].

Hardware/software partitioning problem has a first level impact on system's performance because a wrong choice can result in an increased development time and a final product out of specification. A good partitioning that fit all implementation requirements optimizes operation and minimizes costs usually resulting on an efficient project.

Firstly, partitioning problem was solved by developers with their experience and previously released works. In case of complex systems, this work became arduous and this fact inducted this research field interest on automating the process using techniques like evolutionary computing and artificial neural networks [7]. This automation can be achieved with EDA (electronic design automation) tools or methods that allow a fast search for good solutions on search space. Among existing methods, the profiling-based methods are being widely used nowadays.

3.1 Profiling-Based Methods

There are many methods for hardware/software co-design and the most commonly used nowadays are the profiling-based methods [14].

The increasing complexity of codes raised the need for profiling tools. Profiling tools make code analysis and indicate several system features like functions execution time and memory usage. Each tool gives a specific group of information and most of them are based on clock cycle information. Table 1 presents some other relevant information about profiling tools [14].

Table 1. Profiling Tool's Comparison.

	Memory	Power	Per Funct	Per Line	Call Grph
Gprof	-	-	x	x	x
HDL Profiling	x	x	-	-	-
ATOMIUM	x	-	x	-	x
MEMTRACE	x	x	x	x	-

After obtaining profiling information, system can be modified to achieve expected performance. Usually the critical parts of the system (intensive processing and time consuming routines) start been optimized with by modifications and improvements, then followed by hardware development on extreme cases. This is a cyclic refinement process and usually stops when co-design performance constraints or maximum time-to-market are achieved.

One of the most important features of this kind of method is that the critical part of the co-design, hardware/software partitioning, is performed during a practical refinement process. Due to this fact there is a high probability that the final system has one of the best system partitioning of the search space. Nowadays hardware/software co-design is using soft-processors because they allow software execution, and also designed hardware to be included as custom instructions for validation.

4 Method

Previous sections presented this work's development context. Considering this, the main goal is to implement normalized cross-correlation coefficient calculus in an embedded platform and achieve acceptable execution time using a profile-based hardware/software co-design development method.

4.1 NCC Coefficient Calculus

The computational cost of the cross-correlation coefficient calculus without normalization is lower then normalized, but the results are sensible to image and sub-image brightness and scale changes [10]. So the normalization of this coefficient solves these two important image processing problems. As presented in Section 2 there are other ways to calculate cross-correlation coefficient but they have disadvantages for hardware implementation. Presented facts justified the choice for NCC algorithm implementation.

$$\gamma(s,t) = \frac{\sum\sum[f(x,y) - \bar{f}]\sum\sum[w(x-s,y-t) - \bar{w}]}{\{\sum\sum[f\,(x,y) - \bar{f}]^2\sum\sum[w(x-s,y-t) - \bar{w}]^2\}^{\frac{1}{2}}} \tag{1}$$

Equation 1 presents the normalized cross-correlation coefficient. f represents an

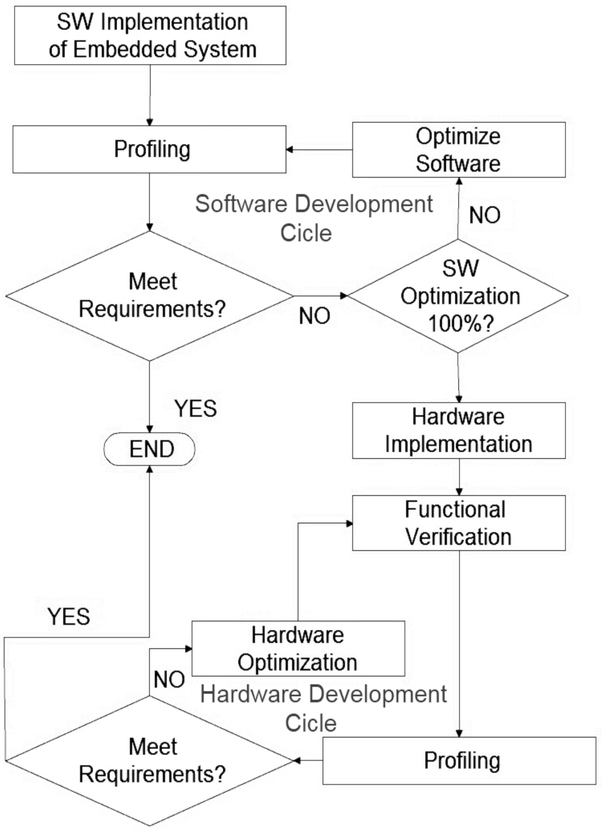

Fig. 1. Modified Profile-Based Method.

image of $M\ x\ N$ size, w represents a sub-image of $m\ x\ n$ where $m \leq M$ and $n \leq N$. Values s and t represents the position of the sub-image center. In this equation, $\bar{w}$ and $\bar{f}$ are respectively the mean value of the sub-image (that is calculated only once) and the mean value of the image region where the sub-image is placed. In this case of template matching $\bar{w}$ refers to what region has to be discovered on in image $\bar{f}$.

4.2 Method Implementation

NCC algorithm was implemented and testes on a general purpose processor (Intel Core i3). After that the proposed hardware/software co-design method (figure 1) was applied. This previous implementation allowed the evaluation of image and sub-image size changes impacts on coefficient calculus. Considering that Nios II soft-processor can be programmed in C language, this procedure will generate a program that can be used on proposed method's first step without increasing development time.

It's important to know the effects of larger images and sub-images in the total computational cost, so general purpose processors were used to evaluate the execution times before porting the program to the soft-processor. This analysis will show what sizes of images run considerably fast on a general purpose processor and these sizes will be the references for soft-processor's algorithm evaluation. General purpose processor implementation allow also software optimization evaluation before being used on the soft-processor.

After this initial step, the proposed co-design method will be applied in order to select the configuration of the soft-processor, aiming to fit system constraints as area consumption, execution time and development time. Selected soft-processor will receive modifications on the software and hardware components according to profile results aiming better execution time. This method is interesting because partitioning, that is considered one of the most important parts of hardware/software co-design, is done implicitly during method execution.

Figure 1 illustrates the flowchart of proposed profile-based method. There are two main cycles were the first one refers to software development and the second one to hardware development. Sometimes system requirements can be satisfied only doing software optimizations. When software optimizations reach the limit without satisfying requirements, hardware development starts. On embedded systems design hardware development is a costly task comparing to software development, so a large amount of time can be saved choosing this modified method. Together with this fact the final solution can be more interesting considering cost, performance and energy consumption because only the portion of the system that needs acceleration will be implemented in hardware.

Basically this method needs the definition of two important things: the profile tool and the soft-processor. In this project the profile tool used was GNU Profiler [11] and the soft-processor was Altera Nios II. The Altera's Nios II [6] soft-processor has a lot of interesting characteristics and the most importants are: (i) the processor can be programmed on C programming language, using tools like GCC compiler and Grpof; (ii) there are three different ways to include custom-designed hardware into the FPGA-based system [16]; (iii)we have two different development platforms available in our lab based on Altera FPGAs (Terasic DE2-70 and Arrow BeMicro).

Nios II has three basic soft-processor configurations: (i) economic (e); (ii) fast (f) and (iii) standard (s). Soft-processors for this work had the same basic configuration with the CPU, on-chip memory, two timers (system timer for clock and another one), JTAG interface for communication. The BeMicro needs extra RAM configured in the FPGA for data storage and a PLL 50 MHz clock and other small features. The main difference between these three processors is that Nios II /e doesn't have instruction pipeline and embedded multipliers, and Nios II /f has hardware improvements on execution time like dedicated hardware for functions and acceleration between pipeline stages. None of them have floating-point unit. It's important to know that these results, except for execution times, are valid for any other soft-processor.

Altera provides softwares for hardware configuration (Quartus II IDE) and for code development and interface with Nios II soft-processor (Nios II EDS). These two softwares have free versions that were used for this work. Developed software will be executed on all different soft-processors to compare their performance, to choose the best between them for this work, and continue the co-design development method with hardware improvements if necessary.

5 Results

Following proposed method, initial results are about the influence of image size and sub-image size changes on execution time for the algorithm. These results are presented graphically on Fig. 2 and Fig. 3. Sub-image sizes are represented by the number of pixels as 10x10, 20x20 and so on, and images are the common sizes of images that are 352x288, 256x192 to 1024x768. These execution image sizes were chosen based on characteristics presented in [4].

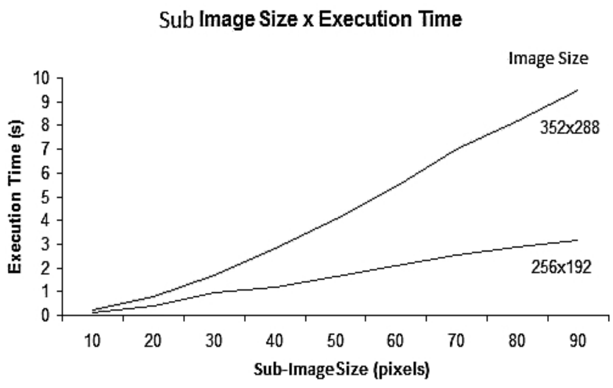

Fig. 2. Sub-Image Size Results for General Purpose Processor.

Based on [30], chosen optimization techniques applied to softwares developed for embedded system are: use of unsigned variables, change functions for equivalent expressions (change pow(x,2) for x*x), use of common operators for simple attributions (change x += for $x = x$ +), loop unrolling and use of vectors instead of matrices.

Results show that the only technique that is really efficient in this case is the loop unrolling. The reason is the fact that compilers apply the first three optimization techniques automatically and the last one is only effective when using operating systems that store matrix data away from each other. The problem of using loop unrolling technique is that the algorithm became more specific

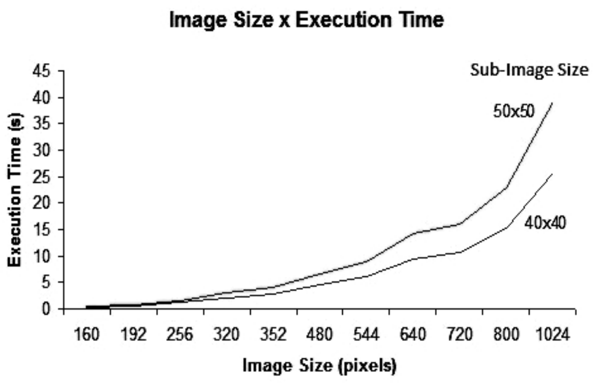

Fig. 3. Image Results for General Purpose Processor.

Table 2. Software Optimization Results

	Mean Execution Time (s)	Size (KB)
Initial	0,047	25,6
Unsigned Variables	0,047	25,6
Function Changing	0,047	25,6
Operator Changing	0,047	25,6
Loop Unrolling	0,0235	26,6

as long as loop unrolling is applied. Other important things are that there is a maximum number of iterations to unroll after that point the execution time became higher again, and the code size also increases. In our tests the maximum number of unrolled iterations was 4.

To evaluate the influence of soft-processor features changing and achieve good results on co-design methodology, the experiments have been done with nine different Nios II hardware configuration in two different FPGAs platforms (Terasic DE2-70 and Arrow BeMicro). Both FPGAs executed the algorithm that implements normalized cross-correlation coefficient described in previous sections.

Starting with pipelined processor influence on execution time and area consumption, three basic processors were configured initially: Nios II /e, /s and /f. This work considers Nios II /e as the reference software implementation of the algorithm because it has no hardware accelerations or improvements as instruction pipelining, hardware multipliers or divisors. After profiling the code the parts that consumed the higher percentage of execution time were the floating point operations. So, the other six possible configurations for Nios II soft processor were implemented using floating point unit (FPU) and hardware division.

Table 3. Comparing Developed Hardware

	*b*ET (s)	*b*LE	*b*MB	*b*P	Clock	*t*ET (s)	*t*LE	*t*MB	*t*P
Nios II /e	5600	14%	53%	30%	50	-	3	72%	<1%
Nios II /s	1260	22%	60%	30%	50	-	4	75%	<1%
Nios II /f	980	26%	63%	30%	50	-	4	77%	<1%
Nios II /ef	4057	22%	53%	30%	50	-	4	72%	<1%
Nios II /sf	980	31%	60%	30%	50	-	6	75%	<1%
Nios II /ff	840	35%	63%	30%	50	-	7	77%	<1%
Nios II /efd	3888	53%	53%	30%	50	-	11	72%	<1%
Nios II /sfd	-	-	-	-	50	856,7	13	75%	<1%
Nios II /ffd	-	-	-	-	50	673,5	14	77%	<1%

FPGAs chosen for this work are Cyclone II and Cyclone III manufactured by Altera Corporation and part of DE2-70 and Arrow BeMicro development boards respectively. These boards are interesting because DE2-70 has a considerable number of I/O and BeMicro is similar to a pen-drive where power is provided from USB connection that consumes a slow amount of power. Only Cyclone II allowed floating point unit configuration with hardware division for Nios II /f and Nios II /s. Fitter of Quartus II wasn't able to fit the complete design (FPU + division) on Cyclone III FPGA so the software could be executed only on seven different soft-processors.

Table 3 shows the results for the execution on all configured soft-processors (Nios II /xf - represents FPU, Nios II /xfd - represents FPU+HW Divisor) and boards (indicating a *b* before - BeMicro board and a *t* before - DE2-70 board) for a 160x120 pixel image and a sub-image of 10x10 pixels (Execution Time (ET), Logic Elements (LE), Pins (P), Memory Bits (MB), Clock).

Execution time using was significantly reduced after including FPU unit without division hardware and on Nios II /e with division hardware. In the other hand, the execution time using DE2-70 was also significantly reduced after including FPU unit, but in this case including the division hardware in all configurations. Based on this results the soft processor Nios II /f with FPU (BeMicro) and Nios II /f with FPU + Division Hardware (DE2-70) were chosen to execute some different sizes of images and templates. Some higher image sizes that execute very fast on the general purpose computer achieve non-acceptable time on Nios II for this implementation as can be seen on table 3. To compare the influence of sizes on the soft-processor some smaller images and sub-images where chosen from 9x9 to 200x200 pixels images and from 3x3 to 11x11 pixels sub-images. These results can be seen in Fig. 5 and Fig. 4.

During hardware optimization step of the proposed methodology, the speedup of the resulting hardwares was calculated. Graphic of Fig.6 shows the speedup values for each development step.

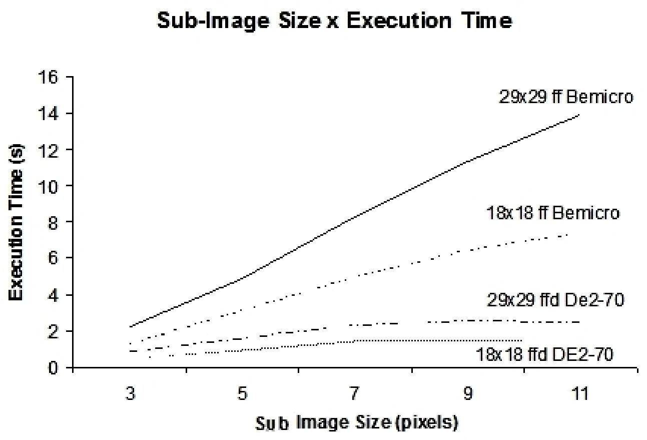

Fig. 4. Sub-Image Results for Nios II Processors.

6 Conclusions

The results presented on previous section show that the proposed architecture achieves acceptable execution time only for small-size images. Initially chosen FPGA, Cyclone III wasn't huge enough to configure Nios II /f with floating point unit and hardware division. Despite of it, if small images can be used this system is very interesting for energy consumption constraints. Development time using chosen IDE's was also interesting because the configuration of the soft-processor became a less arduous task. On DE2-70 the results for the same images were better, representing almost half of the execution time on Bemicro. This improvement occurred because of the division hardware added into floating point unit on co-design hardware development phase. Presented results showed that chosen method is efficient and effective.

Extending this analysis, the speedup rate achieved after hardware development was 8,31. This number is related to all software execution time on Nios II /e processor, equivalent to 5600 seconds, divided by Nios II /f with FPU and hardware division 673,5 seconds. This speedup is due to FPU hardware together with pipeline of 6 stages from Nios II fast processor, and with hardware acceleration for integer multiplies and division. All these modifications can be considered hardware development and can be applied to any soft-processor.

Compilers already do automatically some software optimization operations and this fact justifies the fact that only with loop unrolling technique achieved significant code optimizations as presented on table 2. Using vectors instead of matrices are a good technique only when the operating system doesn't allows a better structured representation of the matrix.

The analysis of image sizes and sub-image sizes impacts on execution time showed that, on general purpose computers with high processing power, grayscale

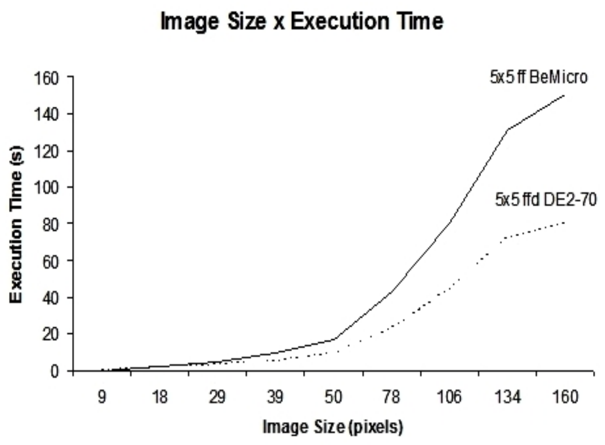

Fig. 5. Image Results for Nios II Processors.

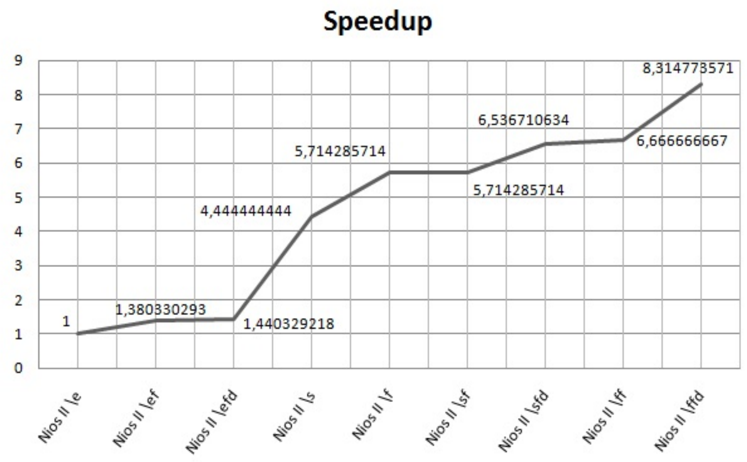

Fig. 6. Speedup during development.

images of 320x240 can achieve acceptable execution time with small sub-images (Fig. 2). This analysis also indicates that the main relative problem is not the image size for the execution time but the sub-image size. There is a point that the sub-image gets so large that the number of calculus is small and execution time start to fall again, due to the reduced sliding window area.Software implementation on soft-processor showed to be more sensitive for both cases related to image sizes, but sub-image size still remains the main problem (Fig. 5).

The comparison between implemented soft-processors also confirms that the usage of the processor with higher computational power (Nios /f + FPU + Hardware Divison) is the right choice for this work and the area that is necessary to configure it on an FPGA compared to simpler processors is an acceptable counterpart. In order to implement this system in an actual autonomous robot navigation system, it will be possible only with improvements on algorithm's limitations related to image sizes that can be executed on acceptable time.

Future works directions include performing tests on larger FPGA's, so the Nios II soft-processor can be implemented with better hardware accelerations and the algorithm can probably achieve more interesting execution times. Other soft-processors can also be tested and compared. Code profile shows that the main problems of this algorithm are the floating point operations, so hardware structures should be included to parallelize code execution of sums and compensate part of the time wasted on divisions and square root operations.

References

1. Atasu, K.: Hadware/Sotware Partitioning for Custom Instruction Processors. Ph.D. thesis, Insitute for Graduate Studies in Science and Engineering (2007)
2. Birla, M., Vikram, K.N.: Partial run-time reconfiguration of fpga for computer vision applications. In: IPDPS. pp. 1–6 (2008)
3. Bobda, C.: Introduction to Reconfigurable Computing: Architectures, Algorithms, and Applications. Springer Publishing Company, Incorporated (2007)
4. Bourque, P.: Image dimensions. http://local.wasp.uwa.edu.au/.../imagedim/ (2010), acesso: 25/10/2010
5. Briechle, K., Hanebeck, U.D.: Template Matching Using Fast Normalized Cross Correlation. In: Proceedings of SPIE: Optical Pattern Recognition XII. vol. 4387, pp. 95–102 (March 2001), `http://dx.doi.org/10.1117/12.421129`
6. Corp., A.: Altera.com. http://www.altera.com/literature (2010), acesso: 22/05/2010
7. Dias, M., Lacerda, W.: Hardware/software co-design using artificial neural network and evolutionaryy computing. In: 5th SPL. pp. 153–157 (April 2009)
8. Feng Zhao, Qingming Huang, W.G.: Image matching by normalized cross-correlation. In: Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on. pp. II 729 – II 732 (2006)
9. Fernandes, C.W., Bellar, M.D., Werneck, M.M.: Cross-correlation-based optical flowmeter. IEEE T. Instrumentation and Measurement 59(4), 840–846 (2010)
10. Gonzalez, R.C., Woods, R.E.: Digital Image Processing (3rd Edition). Prentice-Hall, Inc., Upper Saddle River, NJ, USA (2006)
11. Honeyford, M.: Speed your code with the gnu profiler. http://www.ibm.com/developerworks/library/l-gnuprof.html (2010), acesso: 17/07/2010
12. Hongcheng, Y., Liu, W.: Design of time difference of arrival estimation system based on fast cross correlation. In: Proceedings of 2nd International Conference on Future Computer and Communication. vol. 2, pp. 464–466. IEEE Computer Society (2010)
13. Houssain, S.T.: An introduction to evolutionary computation (1998)

14. Hübert, H., Stabernack, B.: Profiling-based hardware/software co-exploration for the design of video coding architectures. IEEE Trans. Cir. and Sys. for Video Technol. 19(11), 1680–1691 (2009)
15. Jones, S., Andresen, C., Crowley, J.: Appearance based process for visual navigation. In: Intelligent Robots and Systems, 1997. IROS '97., Proceedings of the 1997 IEEE/RSJ International Conference on. vol. 2, pp. 551–557 vol.2 (Sep 1997)
16. Joost, R., Salomon, R.: Hardware-software co-design in practice: A case study. In: in Image Processing, In Proceedings of the 32 nd Annual Conference of the IEEE Industrial Electronics Society (IECON (2006)
17. Kalomiros, J., Lygouras, J.: A reconfigurable architecture for stereo-assisted detection of point-features for robot mapping (2009)
18. Lee, G.G., Chen, Y.K., Mattavelli, M., Jang, E.: Algorithm/architecture co-exploration of visual computing on emergent platforms: Overview and future prospects. Circuits and Systems for Video Technology, IEEE Transactions on 19(11), 1576–1587 (Nov 2009)
19. Lu, J., Rogmans, S., Lafruit, G., Catthoor, F.: Stream-centric stereo matching and view synthesis: A high-speed approach on gpus. Circuits and Systems for Video Technology, IEEE Transactions on 19(11), 1598–1611 (Nov 2009)
20. Matsumoto, Y., Inaba, M., Inoue, H.: Visual navigation using view-sequenced route representation. In: Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on. vol. 1, pp. 83–88 (Apr 1996)
21. de Micheli, G., Gupta, R.K.: Hardware/software co-design. In: Proceedings of IEEE, vol. 85. pp. 349–365 (1997)
22. Nicosevici, T., Garcia, R.: On-line visual vocabularies for robot navigation and mapping. In: Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on. pp. 205–212 (Oct 2009)
23. Pei, L., Xie, Z., Dai, J.: Fast normalized cross-correlation image matching based on multiscale edge information. In: Proceedings of International Conference on Computer Application and System Modeling. vol. 2, pp. 507–511. IEEE Computer Society (2010)
24. Russ, J.C.: Image Processing Handbook, Fourth Edition. CRC Press, Inc., Boca Raton, FL, USA (2002)
25. Sonka, M., Hlavac, V., Boyle, R.: Image Processing, Analysis, and Machine Vision. Thomson-Engineering (2007)
26. Straunstrup, J., Wolf, W.: Hardware Software Co-Design : Principles and Parctice. Springer, first edn. (1997)
27. Tao, Z., Feng-ping, Y., Hao-jun, Q.: An optimized high-speed high-accuracy image matching system based on fpga. In: Proceedings of the International Conference on Information and Automation. pp. 1107–1112. IEEE Computer Society (2010)
28. Tsai, D.M., Lin, C.T.: Fast normalized cross correlation for defect detection. Pattern Recogn. Lett. 24, 2625–2631 (November 2003), http://dx.doi.org/10.1016/S0167-8655(03)00106-5
29. Wolf, W.: A decade of hardware/software codesign. Computer 36(4), 38–43 (April 2003)
30. Wolf, W.: High-Performance Embedded Computing: Architectures, Applications, and Methodologies. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2006)
31. Yiannacouras, P., Rose, J., Steffan, J.G.: The microarchitecture of fpga-based soft processors. In: CASES '05: Proceedings of the 2005 int. conf. on Compilers, arch. and synthesis for emb. sys. pp. 202–212. ACM, New York, NY, USA (2005)

32. Yonghong, Z.: An nprod algorithm ip design for real-time image matching application onto fpga. Electrical and Control Engineering, International Conference on 0, 404–409 (2010)