Hardware/Software Co-design for Real Time Embedded Image Processing: A Case Study

Sol Pedre[1], Tomáš Krajník[2], Elías Todorovich[3], and Patricia Borensztejn[1]

[1] Departamento de Computación, FCEN-UBA, Argentina
{spedre,patricia}@dc.uba.ar
[2] Czech Technical University in Prague, Czech Republic
tkrajnik@labe.felk.cvut.cz
[3] Departamento de Computación y Sistemas, FCE-UNICEN, Argentina
etodorov@exa.unicen.edu.ar

Abstract. Many image processing applications need real time performance, while having restrictions of size, weight and power consumption. These include a wide range of embedded systems from remote sensing applications to mobile phones. FPGA-based solutions are common for these applications, their main drawback being long development time. In this work a co-design methodology for processor-centric embedded systems with hardware acceleration using FPGAs is applied to an image processing method for localization of multiple robots. The goal of the methodology is to achieve a real-time embedded solution using hardware acceleration, but with development time similar to software projects. The final embedded co-designed solution processes 1600×1200 pixel images at a rate of 25 fps, achieving a $12.6\times$ acceleration from the original software solution. This solution runs with a comparable speed as up-to-date PC-based systems, and it is smaller, cheaper and demands less power.

Keywords: real time image processing, hardware/software co-design methodology, FPGA, robotics.

1 Introduction

Many image processing applications require solutions that achieve real time performance. A usual approach for accelerating is to exploit their inherent parallel sections, building implementations on parallel architectures such as GPUs or FPGAs. The appearance of the CUDA [1] framework allowed to implement image processing methods on GPUs (graphical hardware of common PCs) with relative small coding effort resulting in their significant speedup [2, 3]. Although these implementations are based on affordable computational hardware, they are unsuitable for applications that also require small and low power consuming solutions. These cover a wide range of embedded systems from remote sensing applications and robotics to mobile phones and consumer electronics.

An alternative solution is to use Field Programmable Gate Arrays. FPGAs are devices made up of thousands of logic cells and memory. Both the logic cells

L. Alvarez et al. (Eds.): CIARP 2012, LNCS 7441, pp. 599–606, 2012.

and their interconnections are programmable using a standard computer. Their highly parallel architecture with low power consumption, small size and weight provide an excellent platform for achieving real time performance on embedded applications. The inclusion of processor cores embedded in programmable logic has also made FPGAs an excellent platform for hardware/software co-designed solutions. These solutions try to combine the best of both software and hardware worlds, making use of the ease of programming a processor while designing tailored hardware modules for the most time consuming sections of the application. Several authors [4–6] reported successful implementation of image processing algorithms on FPGA-based hardware, including co-designed solutions [7, 8].

The main drawback of FPGA-based methods is time consuming development, that raise exponentially as the complexity of the application increases. This led to the proposal of new methodologies, tools and language aimed at reducing design time by raising the abstraction level of design and implementation. These approaches are commonly known as Electronic System Level (ESL) [9], although they differ in language, abstraction level and tool support.

In this work an image processing case study of a co-design methodology for processor-centric embedded systems with hardware acceleration using FPGAs is presented. The case study is an image processing algorithm for real-time localization and identification of multiple robots, integrated in a remote access robotic laboratory. Results indicate that the proposed methodology is suitable to achieve real-time performance in embedded image processing applications, while reducing the design and coding effort usually needed for this type of tailored co-designed solutions.

2 Methodology Overview

The proposed co-design methodology, described in detail in [10], is aimed at reducing design and coding effort. It has four broad stages: A) Design, B) Implementation and Testing in a general purpose processor, C) Hardware/Software partition and D) Implementation, testing and integration of each hardware module in the final embedded platform.

Taking advantage of the processor centric approach, the whole system is designed using well established high level modeling techniques, languages and tools from the software domain. That is, an Object Oriented Programming (OOP) design approach expressed in Unified Modeling Language (UML) and implemented in C++. This helps to reduce design effort by raising abstraction level while not imposing the need to learn new languages and tools. There are several related works that use domain-specific specializations of UML profiles for hardware or co-design specifications [11] [12] [13]. However, different degrees of hardware details still need to be specified in these approaches. In our approach, the UML design is done prior to hardware/software partition, abstracting away the implementation details related to both hardware and software.

The C++ implementation is then tested in a general purpose processor using debugging resources available in these processors. This implementation not only

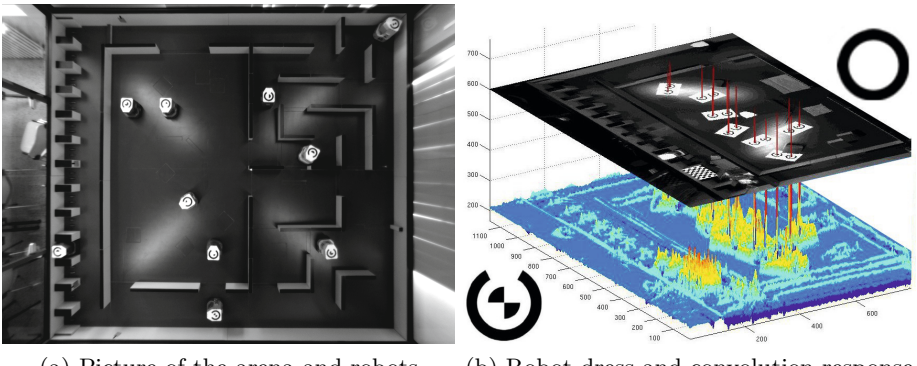
provides a golden reference model, but may also be used as part of the final embedded software. In this manner, software coding effort is reduced.

To perform hardware/software partition, the complete software solution must be migrated to the final embedded processor. The required hardware resources to make the embedded processor run are characterized and the hardware platform is generated. The software platform for the embedded processor is generated and the software solution is migrated. Using profiling tools in the embedded processor, the methods that need to be accelerated by hardware are identified completing the hardware/software partition phase. The modular OOP design facilitates to find the exact methods that need to be accelerated, preventing useless translations to hardware and hence reducing hardware coding effort.

Finally, each hardware module is implemented, tested and integrated in the complete system. Related work in the area of High-Level synthesis include semi-automatic tools to translate code in C to Hardware Description Language (HDL) code for FPGAs [14] [15] [16]. All these require rewriting effort on the original code to particular C language subset, and hardware knowledge in order to generate correct HDL. Our approach is to use the two-process structured VHDL design method [17] for hardware implementation by translating the C++ object methods by hand in a guided way. This method has proven to reduce man-years, code lines and bugs in many major projects [18].

3 Multiple Robot Localization

The System for Robotic Teleeducation (SyRoTek)[19] is a system developed by the Intelligent and Mobile Robotics Group at the Czech Technical University in Prague. This virtual lab provides remote access from anywhere around the world to real mobile robots located in an arena.



(a) Picture of the arena and robots (b) Robot dress and convolution response

Fig. 1. Localization system in SyroTek

In order to perform localization, each robot carries a unique ring-like pattern that allows to calculate its position and orientation in a 1600×1200 gray scale image taken by an overhead camera (see Fig. 1). The image is processed in several steps. First, the radial distortion caused by camera lens imprecision is removed. Then, the image is transformed to make the arena appear as a rectangle aligned with the image edges. Using the intrinsic and extrinsic parameters of the camera, a look-up table mapping pixel coordinates of the rectified image to pixel coordinates of the captured image is computed. Using this look-up table, both transformations are performed in a single step, achieving a faster undistortion. For more accurate results, bilinear interpolation with four surrounding pixels is used to calculate the gray level of the destination pixel.

The rectified image is then convolved with a 40×40 annulus pattern. The maximal values of the convolution filter indicate robot positions on the arena, see Fig.1. Knowing robot positions allows to find the endpoints of the robot dress arc and to determine the robot heading. When orientations are found, each robot is identified by a binary code in the dress center.

The convolution of the entire image is slower than the camera frame rate and therefore it is performed only at system start. After that, the convolution is computed in a neighborhood of each robot's position in the previous frame. Therefore, the correction for image distortion is only performed in those neighborhoods, greatly diminishing the amount of memory accesses needed.

4 Hardware/Software Co-designed Solution

4.1 UML Structural and Behavioral Design

The overall structural design of the solution is shown in the Fig. 2.

The `Robot` class contains the information of each robot, i.e position, heading and id. The class `PositionCalculator` calculates the new position of a robot implementing the image convolution in the `exec` method. The class `AngleCalculator` calculates the new heading of a robot.

A `Matrix` class was created to encapsulate matrix operations. Since most matrices are sub-matrices of bigger ones (e.g. an image section is a sub-matrix of image), memory is only dealt with in very specific moments. The `LoadableMatrix` class inherits from `Matrix` and encapsulates actual memory movements. Finally, the `Image` class is a particular `LoadableMatrix` that knows about image undistortion operation. This class implements both bilinear and nearest neighbor interpolation for comparison purposes.

4.2 Hardware-Software Partition

An Avnet development kit including a Virtex4-FX12 FPGA with a PowerPC405 embedded processor was used. The development tools used are Xilinx's Design Suite 11.2 for hardware and embedded software development and GNU valgrind and gprof for preliminary resource characterization.

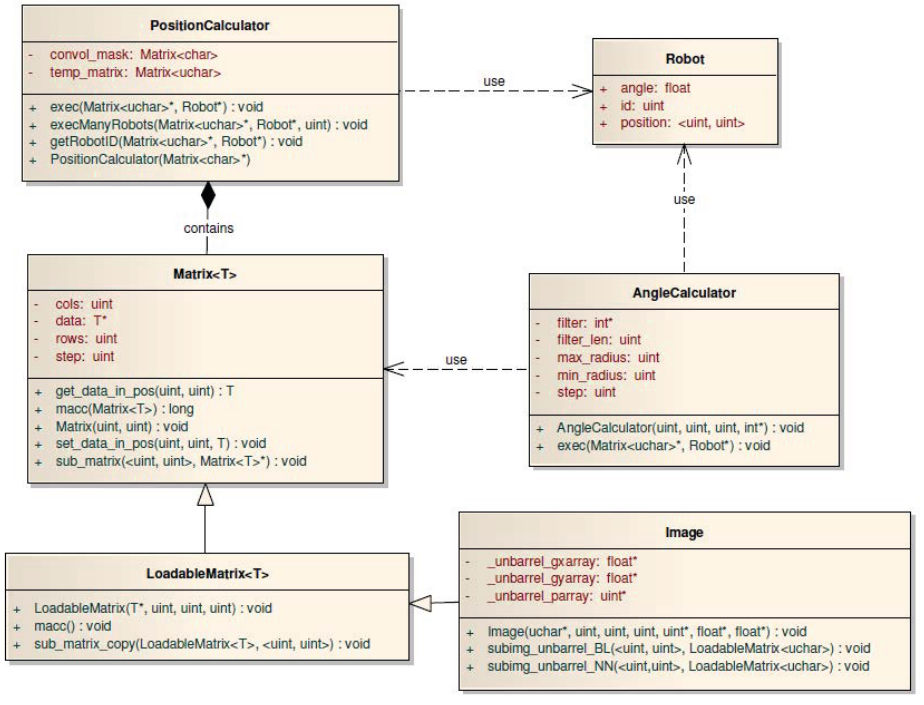


Fig. 2. Structural Design

First, the resources needed to run the software solution in the embedded processor were characterized. Profiling results in a Core i5 M480 (2 cores@2.67GHz) show that the `Matrix::macc` method uses 87% of the time, and is a clear candidate for hardware implementation. This method is called 100 times in `PositionCalculator::exec` that searches for the new position of a robot. The modularity of the OOP design and the encapsulation of the matrix operations in a separate class allows the profiling to accurately point where the most time consuming operation is, preventing useless translations to hardware.

Next, the needed hardware platform to run the software solution in the embedded processor is generated. From the previous analysis, the memories included are Flash, SDRAM and Block RAMs (memory embedded in the FPGA), connected through an IBM PLB bus to the processor. Internal caches were configured. The PowerPC405 is set at its maximum frequency (300 MHz). A standalone (no operating system) software platform was generated for the processor.

The migration of the complete software solution to the embedded processor required only two minor changes. In the embedded solution, images that were opened with OpenCV are loaded from the Flash memory, and dynamic memory for image sections is replaced by BlockRAMs. These interface changes were encapsulated in a single configuration file, so the rest of the code is unchanged.

Finally, the complete software solution is profiled in the embedded processor. Since the PowerPC405 has no FPU, all floating point operations are emulated by Xilinx's compiler. Hence, software optimizations were developed for the PowerPC's particular architecture. Profiling results for each code version are shown in Table 1. The first column corresponds to the original code. The second column uses pre-calculated cosine and sine masks for angle estimation. The third column corresponds to changing all floating point operations to fixed point arithmetics. The fourth column keeps previous changes but performs image undistortion without bilinear interpolation. The fifth column simplifies angle calculation, removing bilinear interpolation for pixel brightness calculation. The last two changes add a 1.6% error in worst case for angle estimation.

All these changes were first implemented and tested in the Corei5, using its debugging and testing resources, and keeping the golden reference model up to date. Migration to the PowerPC did not require code changes. The test suite were images with 14 robots in the arena loaded in the Flash memory. Results for profiling in the Corei5 processor are also shown in this table. The fastest code was used for this test (including all optimizations and floating point arithmetics).

Table 1. Profiling results. All times in miliseconds.

	PPC405@300 MHz					i5@2.67GHz
	orig. code	cos_mask	fixed pt.	unbarrel_NI	angle_NI	all opt.
Matrix::macc	117.7	117.7	117.7	117.7	117.7	28.69
angleCalc::exec	246.9	48.3	17.7	17.7	7.3	0.84
Image::unbarrel	120.0	120.0	120.0	4.5	4.5	0.72
complete solution	*489.5*	*295.6*	*264.0*	*141.7*	*130.2*	*30.74*

An output of this stage is the complete, correct and optimized software version running in the embedded PowerPC405 processor. Also, the definite hardware-software partition is to translate the `Matrix::macc` method to hardware.

4.3 Hardware Implementation, Testing and Integration

Next, the hardware module for the `Matrix::macc` is implemented, including its interface with the memory and embedded processor. Hardware and software changes are introduced to integrate this hardware module in the solution.

The `macc` does the convolution of two matrices. To access data, it is connected to two Block RAMs, one per matrix. The PowerPC is connected to the other port of each Block RAM so it can load data (i.e, the image section and the convolution mask). The `macc` module is also connected by a Device-Control Register(DCR) bus to the PowerPC. This is a simple bus that can connect many slave modules in a daisy chain manner. Through this bus, the PowerPC tells the `macc` module in which address of each Block RAM the matrix to be multiplied starts. When the multiplication is over, the `macc` returns the accumulated value through this DCR bus to the PPC. The six modules and seven interface packages needed for the solution were implemented following the two-process VHDL design method.

In Table 2 a comparison between the complete software solution and the solution with hardware acceleration and software optimizations can be found.

Table 2. Solution comparison in the Virtex4 FPGA

		all software	hard accelerated
Area	Slices	3,575	3,718
	BRAM	13	15
	DSP48	0	1
Time (ms)	Matrix::macc	117.7	22.4
	angleCalc::exec	246.9	7.3
	Image::unbarrel	120.0	4.5
	complete solution	*489.5*	*38.7*

The best possible complete-system performance is achieved since each part (hardware and software) runs at its maximum frequency. For this, a Digital Clock Manager (DCM) is included and the connection between the embedded processor and hardware is done in an asynchronous way (i.e, using memories and the DCR bus).

The final hardware accelerated solution processes 25 fps of 1600×1200 pixel images, achieving a real-time embedded solution for the problem. The acceleration from the original solution to the final software optimized and hardware accelerated solution is 12.6×. The extra FPGA area required is one DSP48 (i.e, a hardware multiplier), 2 Block RAMs and 143 slices, only 4% area penalty. The hardware accelerated solution takes 38.7 ms to process an image while the most optimized software solution in a Corei5 (2 cores@2.67GHz) takes 30.4 ms.

5 Conclusions

The stages of a co-design methodology for processor-centric embedded systems with hardware acceleration using FPGAs were applied to an image processing case study for localization of multiple robots. The aim of the methodology is to achieve a real-time embedded solution using hardware acceleration, but with development times similar to software projects. Results indicate that the proposed methodology is suitable to achieve real-time performance in embedded image processing applications, while reducing design time and coding effort usually needed for this type of tailored co-designed solutions. The achieved embedded solution successfully processes 1600×1200 pixel images at a rate of 25 fps. It runs with a comparable speed as the method implementation on an up-to-date general purpose processor, but is smaller, cheaper and demands less power.

Acknowledgments. Xilinx Design Suite was donated by Xilinx University Program. This work has been partially supported by Czech project No. 7AMB12AR022, and Argentinien projects MINCyT RC/11/20, UBACyT 200158 and PICT-2009-0041.

References

1. NVIDIA: CUDA: Parallel Programming (January 2012), http://www.nvidia.com
2. Jošth, R., et al.: Real-time PCA calculation for spectral imaging (using SIMD and GP-GPU). Journal of Real-Time Image Processing, 1–9 (2012)
3. Cornelis, N., van Gool, L.: Fast scale invariant feature detection and matching on programmable graphics hardware. In: IEEE International Conference on Computer Vision and Pattern Recognition, CVPR, Anchorage Alaska (June 2008)
4. Diaz, J., et al.: FPGA-based real-time optical-flow system. IEEE Transactions on Circuits and Systems for Video Technology 16(2), 274–279 (2006)
5. Pedre, S., Stoliar, A., Borensztejn, P.: Real Time Hot Spot Detection using FPGA. In: 14th Iberoamerican Congress on Pattern Recognition, pp. 595–602. Springer (2009)
6. Bonato, V., Marques, E., Constantinides, G.A.: A Parallel Hardware Architecture for Scale and Rotation Invariant Feature Detection. Transactions on Circuits and Systems for Video Technology 18(12), 1703–1712 (2008)
7. Jordan, H., Dyck, W., Smodic, R.: A co-processed contour tracing algorithm for a smart camera. Journal of RealTime Image Processing 6(1), 23–31 (2010)
8. Castillo, A., Shkvarko, Y., Torres Roman, D., Perez Meana, H.: Convex regularization based hardware/software co-design for real-time enhancement of remote sensing imagery. Journal of Real-Time Image Processing 4, 261–272 (2009)
9. Bailey, B., Martin, G., Piziali, A.: ESL Design and Verification: A prescription for Electronic System-Level Methodology. Morgan Kaufmann (2007)
10. Pedre, S., Krajník, T., Todorovich, E., Borensztejn, P.: A co-design methodology for processor-centric embedded systems with hardware acceleration using FPGA. In: IEEE 8th Southern Programmable Logic Conference, pp. 7–14. IEEE, Brazil (2012)
11. Mallet, F., André, C., DeAntoni, J.: Executing AADL Models with UML/MARTE. In: International Conference of Engineering of Complex Computer Systems, pp. 371–376. IEEE, Germany (2009)
12. Mueller, W., Rosti, A., Bocchio, S., Riccobene, E., Scandurra, P., Dehaene, W., Vanderperren, Y., Ku, L.: UML for ESL Design - Basic Principles, Tools, and Applications. In: IEEE/ACM Int. Conf. on Computer Aided Design, pp. 73–80 (November 2006)
13. Silva-Filho, A.G., et al.: An ESL Approach for Energy Consumption Analysis of Cache Memories in SoC Platforms. International Journal of Reconfigurable Computing, 1–12 (2011)
14. Jacquard: ROCCC 2.0 (October 2011), http://www.jacquardcomputing.com/roccc/
15. Mentor-Graphics: CatapultC (October 2011), http://www.mentor.com/esl/catapult
16. Nallatech: DIME-C (October 2011), www.nallatech.com/Development-Tools/dime-c.html
17. Gaisler, J.: A structured VHDL design method. In: Fault-tolerant Microprocessors for Space Applications. Gaisler Research, pp. 41–50 (2004)
18. ESA: European Space Agency VHDL (October 2011), http://www.esa.int
19. Kulich, M., et al.: SyRoTek - On an e-Learning System for Mobile Robotics and Artificial Intelligence. In: ICAART 2009, pp. 275–280. INSTICC Press, Setúbal (2009)