A Profile-Based Method for Hardware/Software Co-design Applied in Evolutionary Robotics Using Reconfigurable Computing

Mauricio A Dias , Daniel O Sales and Fernando S Osorio
Institute of Mathematical Science and Computation - ICMC
Mobile Robotics Laboratory - University of Sao Paulo - USP
Sao Carlos - Sao Paulo
macdias, dsales, fosorio @ icmc.usp.br

***Abstract*—Evolutionary algorithms are a very common technique on computational intelligence field. Some algorithms need a huge amount of memory, making them not so trivial to apply on embedded systems. In this work a profile-based approach is proposed and applied in an evolutionary algorithm with some characteristics that allow its use on embedded systems: the micro-GA. The main goal is to improve the execution time compared to other software algorithms obtaining also an acceptable design time. The results presents comparisons between implementations and discussions about soft-processor features influence on this type of algorithm.**

***Keywords*-FPGA; Micro Genetic Algorithm; Evolutionary Computation; Hardware Optimization; HW/SW Co-Design**

I. Introduction

In recent years some heuristic methods gained researcher's attention, specially Artificial Neural Networks [1] and Evolutionary Algorithms [2]. One specific implementation of evolutionary algorithm is the Genetic Algorithm (GA), proposed in 70's by Holland and his students, and diffused in 1989 by Goldberg [3]. In this algorithm, the descendants are generated by operations between individuals of the population, and the most important operator is crossover.

The algorithm is divided in two basic steps: create initial population and execute the main loop. The main loop consists of making interactions with individuals (crossover and mutation) and evaluating them, until reach the stopping criteria, which may be a maximum number of generations, or obtaining an individual with determinate fitness value. With this proceeding, the algorithm is able to keep the best solutions and explore the search space at the same time.

Evolutionary algorithms have been presenting good results in many computational fields. Robotics is one of them. Evolutionary robotics is an area that is receiving an increasing attention from roboticss researchers recently. There are many possible applications for this kind of algorithms on robotics field from robot's physical configuration to control systems and navigation [2].

Control systems for autonomous robots are often programmed by researchers or designers. As the complexity of an environment and task for an autonomous robot increases, the difficulty of designing control systems by hand becomes a limiting factor in the degree of functional complexity that can be achieved[2]. One possible solution for this problem is to use an automatic learning method as evolutionary computing.

Robots can be seen as embedded systems or systems composed by a group of embedded systems. In most cases embedded systems have limited capacity of processing and memory, and developing embedded systems for robotics is a complex task because these systems share resources with sensors and actuators. The combination of these factors justifies the usage of more robust techniques for embedded robotic system's design considering performance, costs, energy consumption, processing and execution time.

At first, the solution for embedded systems design was to implement algorithms directly in hardware. Hardware implementation projects nowadays have been replaced by hardware/software co-designs mainly on embedded systems design. This choice is based primarily on increasing complexity of embedded systems, reduction of time-to-market for embedded systems design and increasing availability of hardware with lower costs. With more available hardware designers are more concerned with functionalities than hardware optimization resulting on more successful and complex projects in the same design time.

Co-design methods usually require flexible development tools that allows rapid prototyping. One way to achieve this desired flexibility level is to develop hardware with reconfigurable computing devices such as FPGA's.

Reconfigurable computing can be defined as the study of computing with reconfigurable devices [4]. In essence this definition means: reconfigurable computing is a paradigm that has the flexibility of general purpose processors combined with the performance of application specific integrated circuits. Of course this characteristics are present on reconfigurable computing in less expressive ways but this paradigm tends to achieve high performance with high flexibility one day. To implement reconfigurable computing it's necessary to use reconfigurable devices. Reconfigurable devices are devices that allow the process of changing its structure at run-time. There are various types of reconfigurable devices and one of them is specially interesting for

978-0-7695-4204-1/10 $26.00 © 2010 IEEE
DOI 10.1109/CERMA.2010.59

this work, the FPGA's.

Field Programmable Gate Arrays (FPGA's) are programmable devices consisting of three main parts: programmable logic cells, configurable logic blocks and I/O cells[4]. FPGA's are flexible devices because, among various features, allow hardware to be described using using high-level Hardware Description Languages (HDL's) and several reconfigurations. In this work specifically, there is a free soft-processor provided by FPGA's manufacturer that was 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: a programmable instruction processor implemented in the reconfigurable logic of the FPGA [5]. Key benefits of using a soft processor include configurability to trade between price and performance, faster time to market, easy integration with the FPGA fabric, and avoiding obsolescence. Soft processors have several advantages and a designer can implement the exact number of soft-processor cores required by the application. Since it is implemented in configurable logic, a soft processor can be tuned by varying its implementation and complexity to match the exact requirements of an application [5].

So, this work proposes a profile-based method for hardware/software co-design and presents results of applying the proposed method for a μGA hardware implementation.

II. μ Genetic Algortithm

Micro Genetic Algorithm (μGA) is a genetic algorithm with very small population and simple genetic parameters, used for solving function optimization problems. It was proposed by Krishnakumar in 1989 [6] as a faster alternative to Simple GA [3] and other usual implementations of GA's.

The result achieved in Krishnakumar's work shows that its implementation is quicker than big population approaches in spite of its simplicity. As the number of individuals is small, the time penalty involved in evaluating the fitness functions and performing the genetic operators generation after generation is smaller.

It is known that small populations generally converge to non-optimal results, due to insufficient information processing and low diversity between individuals. This problem is solved transferring only the best individual to a new population, and generating all other individuals randomly avoiding early convergence. In this approach, a small size population is generated randomly, and genetic operations are performed until reach the nominal convergence. Then, the best individual is transferred to a new population and the remaining individuals are generated randomly. So, the algorithm repeat this process until reach the stopping criteria.

Unlike in other GA implementations, the solution can be found by evaluating the fitness of the best individual, not only when all individuals converge to a single value. This is also an advantage which makes μGA finish faster. The crossover rate is usually 1, and mutation rate must be near to 0, because enough diversity is introduced every time the population is re-initialized.

Micro-Genetic Algorithms are useful in many applications, for example Real-Time Systems, and particulary Evolutionary Robotics, where each robot can be represented as one individual, and to have many robots is unfeasible.

III. 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, the technical challenges in co-design will also increase [7]. Engineering systems can be classified as a joined operation component group developed for some task resolution [8]. These components can be implemented in hardware or in software, depending on what are the system's constraints.

Hardware/Software co-design 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 [7]. Hardware/Software co-design means the union of all system goals exploring interaction between hardware and software components during development [9]. Figure 1 presents simplified co-design flow.

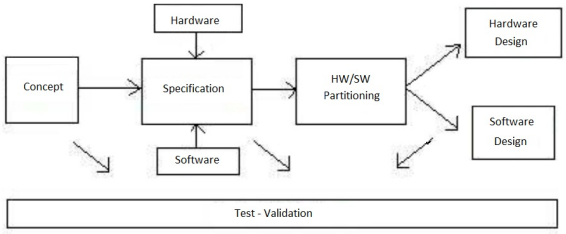

Figure 1. Co-design Flow.

Co-design is a methodology that explores the synergism of hardware and software focusing optimization of interested parts of the system and tries to increase the predictability of the system design by providing method analysis that shows to the designer if the system meets it's performance constraints [10]. This method really improve development in cases that the system is designed jointly,eliminating system's unnecessary components.

Hardware/Software co-design is a hard task and computational tools appeared to make this work faster and increases 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 [11].

Hardware/software partitioning problem has a first level impact on system's performance because a wrong choice can result in an increased develop 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.

Initially, partitioning problems were solved by developers with their experience and previously released works. In case of complex systems, this work became arduous and this fact inducted the interest on automating the process using techniques like evolutionary computing and artificial neural networks[12]. 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 [13].

A. *Profiling-Based Methods*

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

	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	-

Table I
PROFILING TOOL'S COMPARISON [13].

The performance of embedded systems is dominated by accompanying application code[13]. 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 have clock cycle information and table I presents some other relevant information[13].

After obtaining profiling information, system can be modified to achieve expected performance. Profile-based methods propose code modification and improvement 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. Figure 2 illustrates general profile-based method design flow.

One of the most important features of this kind of methods 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 partitioning contained in the search space.

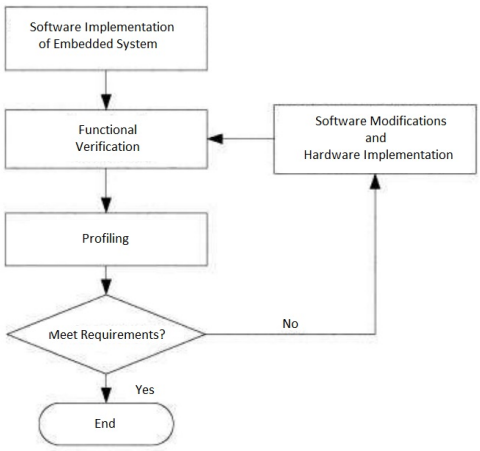

Figure 2. General Profile-Based Method [14].

Nowadays hardware/software co-design is using soft-processors to run developed software and to include designed hardware as custom instructions for testing.

IV. METHOD IMPLEMENTATION

Previous sections presented the context that this work is inserted. Considering these concepts the main idea is to implement μGA in an embedded platform and achieve an acceptable execution time for robotic applications. To validate the implementation, some benchmark functions were tested with many hardware configurations to evaluate area consumption, system clock, execution time and soft-processor pipeline influence.

Analyzing basic profile-based method implementation presented in figure 2, some improvements can be done to reduce co-design time and find a better final system. Figure 3 contains new method's flowchart.

A deep analysis of figure 3 flowchart shows that there is an improvement on development time. There are two main cycles, the first one of software development and the second one of 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 in many characteristics like cost, performance and energy because only the portion of the system that needs acceleration will be implemented in hardware.

Some fitness functions were chosen to evaluate developed systems during profiling-based co-design method. There are

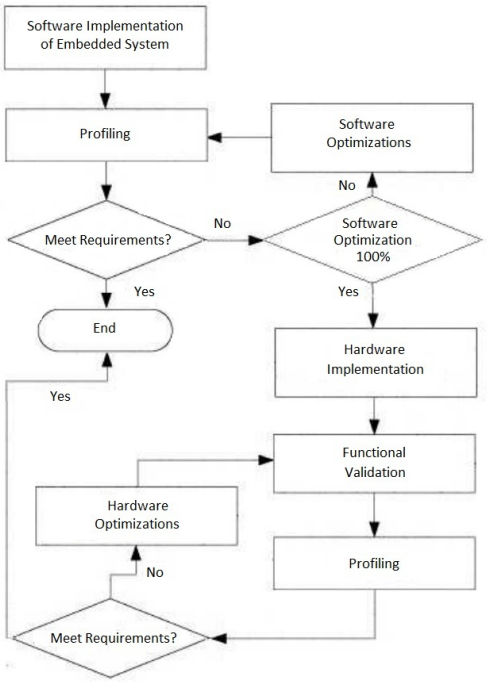

Figure 3. Modified Profile-Based Method.

many benchmark fitness functions available and analyzing functions commonly used on robotics applications presented in [2]. Some benchmark functions based on [15] were selected.

The benchmark functions used to evaluate the performance are some classical benchmark functions found in literature [16],[17]. They are: (f1) Alpine, (f2) sphere, (f3) f6schaffer and (f4) griewank. F6Schaffer is a bi-dimensional function, which has several local minimum. Sphere function is continuous, convex and uni-modal. Alpine and Griewank are multi-modal, and also have several local minimum. Figure 4 shows the plotted graphics of these functions: Alpine (top left), sphere (top right), f6schaffer (bottom left) and griewank (bottom right)[15].

There are many features that can be evaluated on a hardware/software co-design. In this case the most important is execution time, but together with execution time hardware developments should achieve good values on hardware area, maximum system clock, if there is a microprocessor the presence of execution pipeline, dedicated modules and energy consumption. Development time is also important on co-design specially when associated with time-to-market.

Several available development environments and toolboxes could be used in this work's experiments. Due to researches experience, experiments were realized on Altera Quartus II IDE and Nios II EDS and implemented on DE2-70's Cyclone II family FPGA manufactured by Altera. This family is composed by hierarchy-based FPGA's, that means,

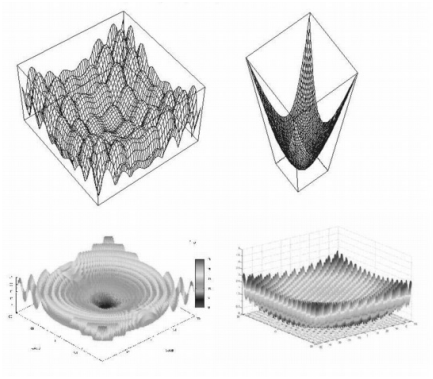

Figure 4. Benchmark Functions.

there are basic logic blocks that are grouped to form larger blocks and so on. Blocks are connected by programmable interconnections and the chip is surrounded by input/output pins. Nios II EDS supports Altera's soft processor Nios II that can be configured on Quartus SOPC Builder. IDE's, information and manuals can be downloaded in [18]. Nios II soft-processor has three types of implementations with different degrees of complexity and supports hardware to be integrated as processor's custom instructions. So developed hardware were included as custom instructions for evaluation. Analyzed profiling were obtained with GProf (Gnu Profiler)[19] that is integrated with Nios II EDS, that has also a gcc compiler for Nios II processors so software development was made on C language.

The stop criteria for all implementations of μGA is the number of generations (f1 - 150, f2 - 200, f3 - 50, f4 - 2000). Stopping these algorithms based on fitness value would cause different execution times and the experiments would not be comparable. So the only difference between algorithms occurs only from function to function not from hardware to hardware and mainly on rates.

V. Results

To evaluate the influence of changing soft-processor features, hardware features and achieve a good result on co-design methodology, the experiments have been done with nine different types of hardware (soft-processor + custom instructions) and each one executing benchmark functions described in section 4. Before doing any hardware development some code modifications have been done. Basically, parameters were adjusted on pc execution of algorithms to reach the minimum number of generations that return the expected results with precision of two decimal places. Before executing on the soft-processor all benchmark function's solution were tested being almost instantaneously executed on a dual-core Intel Pentium.

Experiments were started observing the pipelined processor influence on execution time and area consumption.

	LE	MB	P	Execution Time
Nios II /e	3%	72%	< 1 %	t1
Nios II /s	4%	75%	< 1%	0.15 * t1
Nios II /f	5%	77%	< 1%	0.12 * t1

Table II
PIPELINE AND AREA

Three processors have been configured for these experiments initially, Nios II economic (e), fast (f) and standard (s). Soft-processors had the same basic configuration with the CPU, 100K of on-chip memory, timer for clock and JTAG communication interface. The main difference between these three processors is that Nios II /e doesn't have pipeline and embedded multipliers, and Nios II /f has hardware improvements on execution time like dedicated hardware for functions and acceleration between pipeline stages. It's important to know that these results are valid for any soft-processor, and this specific choice (NIOS II) is related to our research group experience with these particular tools and IDEs. Table 1 shows the results of the first developed hardware considering: area consumption of the FPGA (Logic Elements (LE), Pins (P), Memory Bits (MB)) and the comparison between mean execution time for benchmark functions defined on section 4.

These execution times show that Nios II /f processor has the best time but the higher area consumption. In this case, is interesting to evaluate if a simple pipelined processor executes the software on acceptable time to save some chip area. Execution time difference between non-pipelined processors and pipelined processors is remarkable. Execution profiles showed that, as expected, the main bottleneck of μGA is the fitness function. Crossover and mutation are also problems but 2 orders of magnitude below and this fact proves that the change of parameters does not invalidate comparisons between benchmark functions results.

Benchmark functions used had basically the same characteristic: they work with floating point numbers. There are two basic ways to accelerate floating point based functions: build dedicated hardware to execute exactly that fitness function or uses a floating point unit (FPU). Dedicated hardware will certainly achieve the best execution time. In this case, is necessary to maintain the maximum of flexibility as possible, because these hardwares will be applied in robotic applications and should be able to execute different code, whenever possible, with the best available hardware configuration. Some functions didn't need floating point division hardware but experiments considered the adoption of this hardware to analyze the influence of this hardware presence for all executions.

Firstly, all hardware characteristics were analyzed. Table 3 shows the results of all developed hardware systems (Nios II /xf - FPU, Nios II /xfd - FPU+HW Divisor) area consumption and maximum clock achieved. It's important to say that no compilation optimization has been set on Quartus II IDE to make results the most generical as possible.

	LE	MB	P	Clock(MHz)
Nios II /e	3%	72%	< 1%	108.80
Nios II /s	4%	75%	< 1%	97.54
Nios II /f	5%	77%	< 1%	113.06
Nios II /ef	4%	72%	< 1%	106.26
Nios II /sf	6%	75%	< 1%	89.66
Nios II /ff	7%	77%	< 1%	116.21
Nios II /efd	11%	72%	< 1%	107.70
Nios II /sfd	13%	75%	< 1%	93.39
Nios II /ffd	14%	77%	< 1%	113.34

Table III
COMPARING DEVELOPED HARDWARE

	Alpine	Sphere	F6schaffer	Griewank
Nios II /e	33.66	2.60	4.01	1404.23
Nios II /s	3.90	0.39	0.52	363.26
Nios II /f	3.04	0.31	0.41	273.32
Nios II /ef	28.28	1.59	3.48	1419.64
Nios II /sf	3.24	0.24	0.47	351.93
Nios II /ff	2.37	0.18	0.36	281.49
Nios II /efd	28.69	1.56	3.41	1367.09
Nios II /sfd	3.28	0.21	0.48	338.54
Nios II /ffd	2.51	0.18	0.39	248.84

Table IV
ALL EXECUTION TIMES

Table 3 contains important information. The chosen profiling tool doesn't have energy consumption information but with this results is possible to find indicatives of this feature. LUT function implementations on FPGA's have the problem that sometimes structures are not totally used and even partially used, consuming the same amount of energy of a fully utilized structure. Other thing that impacts directly on energy consumption is the clock frequency. Then analyzing all these features, in all cases standard processor balanced area consumption with clock frequency. Other thing to notice is that FPU + HW Divisor increases significantly logic element's allocations so execution time has to be analyzed to show whether is justifiable to use this hardware.

After hardware development, execution time (in seconds) for benchmark functions are presented on table 4.

All these execution times are the result of GProf flat profile on Nios II IDE. These numbers present some interesting results. Functions that doesn't need floating point hardware division had an increase on their execution time when it was included. Soft-processor without pipeline presented the worst times compared with pipelined processors. With clock values of table 3, execution times of the first three functions are acceptable for both pipelined processors and the difference between them was minimal.

VI. CONCLUSIONS

Results presented in previous section's tables allowed advantages and disadvantages analysis of each hardware struc-

ture developed in this work. The results of table 3 are ordered by co-design phase. Proposed methodology achieve great results on each phase: first with software development part when codes reached instantaneous execution time on PC, second with pipeline inclusion on a simple soft-processor then optimizing this pipeline and third with FPU inclusion as custom instruction of soft-processor. All implementations final execution time presented a great speedup compared to all-software soft-processor implementation.

μGA proved to be a great alternative to embedded systems because of the small amount of memory spent and good reached results. Hardware solution with soft-processors achieved acceptable execution time for simple and medium-complexity functions. Complex functions like Griewank need some specific hardware development to achieve acceptable execution time to be embedded in a robot system.

As future works, some benchmark functions use trigonometric functions inside them and these functions can be implemented in hardware as tables, decreasing execution time. It is also possible to develop specific hardware for complex fitness functions in order to achieve acceptable execution time and embed this developed hardware on a real robot system.

REFERENCES

[1] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.

[2] A. L. Nelson, G. J. Barlow, and L. Doitsidis, "Fitness functions in evolutionary robotics: A survey and analysis," *Robot. Auton. Syst.*, vol. 57, no. 4, pp. 345–370, 2009.

[3] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1st ed. Addison-Wesley Professional, January 1989.

[4] C. Bobda, *Introduction to Reconfigurable Computing: Architectures, Algorithms, and Applications*. Springer Publishing Company, Incorporated, 2007.

[5] P. Yiannacouras, J. Rose, and J. G. Steffan, "The microarchitecture of fpga-based soft processors," in *CASES '05: Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*. New York, NY, USA: ACM, 2005, pp. 202–212.

[6] K. KRISHNAKUMAR, "Micro-genetic algorithms for stationary and non-stationary function optimization," in *SPIE: Intelligent Control and Adaptive Systems*, 1989, pp. 289–296.

[7] W. Wolf, "A decade of hardware/software codesign," *Computer*, vol. 36, no. 4, pp. 38–43, April 2003.

[8] S. T. Houssain, "An introduction to evolutionary computation," 1998.

[9] J. Straunstrup and W. Wolf, *Hardware Software Co-Design : Principles and Parctice*, 1st ed. Springer, 1997.

[10] A. C. Atoche and D. T. Roman, "Hardware/software codesign for near real time enhancement of remote sensing imaging," in *5th International Conference on Eletrical Engeneering, Computing Science and Automatic Control*. IEEE Computer Society, 2008, pp. 525–530.

[11] G. de Micheli and R. K. Gupta, "Hardware/software co-design," in *Proceedings of IEEE, vol. 85*, 1997, pp. 349–365.

[12] M. Dias and W. Lacerda, "Hardware/software co-design using artificial neural network and evolutionaryy computing," in *Programmable Logic, 2009. SPL. 5th Southern Conference on*, April 2009, pp. 153–157.

[13] H. Hübert and B. Stabernack, "Profiling-based hardware/software co-exploration for the design of video coding architectures," *IEEE Trans. Cir. and Sys. for Video Technol.*, vol. 19, no. 11, pp. 1680–1691, 2009.

[14] K. E. E.-D. EI-Sayed M. Saad, Medhat H. A. Awadalla, "Fpga-based software profiler for hardware/software co-design," in *26t h NATIONAL RADIO SCIENCE CONFERENCE (NRSC2009)*, 2009, pp. D14 1–8.

[15] T. K., Y. X., S. P. N., M. C., C. Y. P., C. C. M., and Y. Z., "Benchmark functions for the cec'2008," in *Special Session and Competition on Large Scale Global Optimization*, 2007, pp. 1–18.

[16] A. B. de Souza, "Fundamentos de otimizacao por inteligencia de enxames: uma visao geral," in *Sba Control & Automation*, vol. 20, 2009, pp. 271–304.

[17] R. Tassing, D. Wang, Y. Yang, and G. Zhu, "Gene sorting in differential evolution," in *ISNN 2009: Proceedings of the 6th International Symposium on Neural Networks*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 663–674.

[18] A. Corp., "Altera.com," http://www.altera.com/literature, 2010, acesso: 22/05/2010.

[19] M. Honeyford, "Speed your code with the gnu profiler," http://www.ibm.com/developerworks/library/l-gnuprof.html, 2010, acesso: 04/07/2010.