Multi-Objective Genetic Algorithm Implemented on a STM32F407 Microcontroller

Pedro Henrique de Oliveira Santos*, Gustavo Luís Soares†, Thiago Melo Machado-Coelho‡
Bernardo Augusto Godinho de Oliveira§, Petr Iakovlevitch Ekel¶,
Flávia Magalhães Freitas Ferreira‖, Carlos Augusto Paiva da Silva Martins**

Pontifical Catholic University of Minas Gerais*†§¶‖**
Graduate Program in Electrical Engineering
Belo Horizonte, Minas Gerais, Brazil, 30140-100
pedrosantos7391@gmail.com*, gsoares@pucminas.br†, bernardo.godinho@sga.pucminas.br§,
ekel@pucminas.br¶, flaviamagfreitas@pucminas.br‖, capsm@pucminas.br**

Federal University of Minas Gerais†
Graduate Program in Electrical Engineering
Belo Horizonte, Minas Gerais, Brasil, 31270-901
thmmcoelho@ufmg.br†

***Abstract*—This paper proposes the implementation of a multi-objective genetic algorithm (MOGA) on a low-end microcontroller, applying benchmark functions to evaluate its performance. A general-purpose microprocessor implementation of the same algorithm was used to compare CPU cycles spent for each generation on both platforms. The results show that although the microcontroller is slower than the microprocessor, it is fast enough to be used for solving optimization problems, and has lower cost and lower power consumption than the microprocessor platform.**

***Keywords*—Multi-Objective Genetic Algorithm, Microcontroller, Multi-objective Optimization.**

I. Introduction

Multi-objective optimization (MOO) is the area concerned with solving problems that have two or more conflicting objective functions. The solution of MOO problems consists of finding a set of solutions that are not dominated by other solutions from the considered objective functions. This set is called Pareto front [1]. A possible way for solving MOO problems is the application of evolutionary algorithms (EAs).

Since EAs use a set of solutions, called a population, instead of a single one, they are suitable for solving MOO problems [2]. Furthermore, they are classified as stochastic algorithms, so their chance of falling into local optima rather than the global is significantly lower. The performance of a population can be evaluated by its closeness to the optimal Pareto Front and by the uniformity of its distribution. Genetic algorithms (GA) are some of the most applied methods for solving MOO problems [1].In the literature (for example, [3]–[10]), it is shown that, despite the simplicity of GAs, they provide convincing results.

Although in many situations, it is required to obtain the best possible solution in the shortest time, some applications permit one to use an approach that takes more time to execute in favor of less energy consumption, lower cost and smaller equipment size. Most GAs, however, are implemented to run on general purpose processors, which may not be the best option for those applications, considering that processors are designed to perform several tasks simultaneously. Low-cost hardware with fewer features, such as microcontrollers, often consumes less energy since its processing power is wholly dedicated to accomplishing the task at hand.

Microcontrollers are small, low-cost, and low-power computers that have been designed to perform specific tasks. Their architecture depends on the application for which they have been built. Usually, they consist of a central processing unit (CPU), random access memory (RAM), electrically erasable programmable read-only memory (EEPROM), input and output pins, timers, serial ports, analog-digital converters, among other peripherals. All these modules are built into a single package, which makes them perfect to be used as embedded systems.

Tasks like path planning for robots, drones, and cars are often performed on-the-go and are highly dependent on a battery to power the systems. Furthermore, those devices are restricted in size, weight, and heat dissipation, which makes the use of general purpose processors unsuited for those tasks. Although the performance is usually worse in the use of microcontrollers, some tasks can safely use a suboptimal solution whose obtaining would significantly reduce the execution time. This, combined with its low-cost and low-power consumption, makes the microcontroller a good substitute for a general processor for optimization tasks in embedded systems.

This paper proposes a microcontroller implementation of multi-objective genetic algorithms (MOGAs) for solving MOO problems without the use of a fully-featured computer or a high-end platform, thus reducing the cost and power consumption. The performance of the MOGAs

978-1-5090-6017-7/18/$31.00 ©2018 IEEE

within a microcontroller will be then compared with a MOGA running on a general purpose computer.

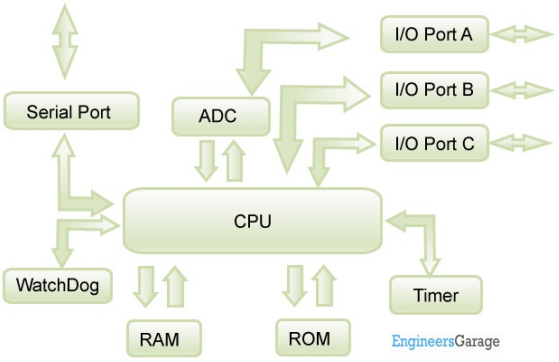
The paper is organized as follows: Section II covers background knowledge about the difference between microcontrollers/microprocessors, GA and works related to this paper; Section III presents the hardware and a MOGA method; Section IV describes the experiments and results. Finally, Section V presents a short conclusion of the work.

II. Background and Related Works

A. Microprocessors and microcontrollers

Typically, EAs are executed on microprocessors inside a general purpose computer, which is different from the microcontroller considered in this work. Although microprocessors and microcontrollers share many features and characteristics, they have notable differences. A microprocessor is designed to perform a comprehensible number of applications. It incorporates only a CPU within its encapsulation. It has no RAM, read-only memory (ROM) or any other peripherals. Besides, these modules must be connected externally to it using a motherboard to be functional. The microcontroller, however, has the CPU and other peripherals on a single chip, as presented in Fig.1, and is designed to perform tasks in which the relationship between input and output are well-defined [11]. The main advantage of using a microcontroller is that it is a dedicated system that consumes less memory and processing power, which makes its cost and power consumption lower than in microprocessors [12].

Fig. 1. Internal Architecture of a Microcontroller [12]



B. Genetic Algorithms

GAs are the most commonly used class of EAs [1]. The typical structure of a GA is presented on Tab. I. In the use of the GA, each solution is expressed as an individual, whereas a group of individuals is called a population. The algorithm begins by initializing the generation counter (step 1) and creating the first population (step 2). The performance of each individual is evaluated in step 3 through the objective function. Steps 4 to 10 select the individuals that will be used in the next generation. The selection method tries to preserve individuals considered most suitable, while the crossover method spreads generic information and, finally, the mutation method applies a diversification operator. The new population is generated, and its performance is calculated. These steps are repeated until a conclusion criterion, such as a maximum number of iterations, is reached.

TABLE I. Typical Genetic Algorithm Structure [1]

inputs(*parameters*)
outputs(*population*)

1 initialize t with 0
2 initialize the *population* with *parameters*
3 compute the performance of the *population*
4 if stop criteria is reached, jump to 11
5 increment t
6 apply *selection* method
7 apply *crossover* method
8 apply *mutation* method
9 compute the performance of the new *population*
10 jump to 4
11 return current *population*

C. Related Works

Several studies compare EAs implemented on microprocessors and implemented on embedded systems, such as microcontrollers, field programmable gate array (FPGAs) or other hardware technologies. Aporntewan and Chongstitvatana [3] have used a 20MHz Xilinx FPGA to implement a Compact Genetic Algorithm (CGA) and its performance has been compared with the same algorithm running on software on a Solaris system with Ultra Sparc II 200 MHz processor. The authors have shown that the FPGA used to perform the algorithm was able to execute it 1,000 times faster than the software implementation. The CGA was chosen because a generic GA was not suitable for hardware implementation so that the work is carried out in the required time.

Valdez et al. [13] have used a genetic programming technique for predicting the behavior of a radio frequency power amplifier. The genetic programming approach has been improved with a local search heuristic to reduce the computational cost and improve the convergence. The results have been compared with a memory polynomial model and showed that the approach is capable not only for solving the proposed task but also yields more reliable results than the memory polynomial model using different metrics.

According to Gallagher et al. [4] CGA is a candidate for hardware implementation because, for various evolutionary hardware applications, the amount of used memory and the energy efficiency are critical aspects of the project, requiring them to be as small as possible. However, CGA has a lower search power. Gallagher et al. [4] proposed changes to the CGA algorithm to improve its search capability without substantially increasing the size and complexity of its implementation in hardware have been compared with an application in C, performed on an Ultra Sparc II processor. The results have shown that similar to Aporntewan and Chongstitvatana [3], the FPGA can run the algorithm up to 1,000 times faster than software.

In the literature, there are examples of GAs running in microcontrollers. For instance, the results of [5] are associated with a microcontroller with extremely limited resources (an Intel 8051 containing 4 Kbytes internal ROM, 128 bytes of internal RAM and 128 Kbytes integrated into external memory) able to run a GA to solve optimization problems and decision making processes for an automated guided vehicle (AVG). The results of [5] have shown that the 8051 was able to perform functions required by AVG, such as path planning, which was made possible by the GA.

Another use of GAs in microcontrollers has directed at optimizing the membership functions, rules, and gains of fuzzy logic in proportional and derivative fuzzy logic controllers (PD-FLC) at control systems. Khan et al. [6] have proposed to implement a GA on a PIC16F877A microcontroller to eliminate problems related to parameters dependency and to add the benefits of simple modifications to a temperature control system to change their operating conditions.

Dendaluce et al. [7] have used a high-performance real-time processor (ARM Cortex-R4F - 180 MHz - 3 MB of flash memory - 256 KB RAM) to implement the non-dominated sorting genetic algorithm II (NSGA-II). This type of hardware is used when fault tolerance, maintainability, and deterministic real-time responses are essential [14]. A neural network (NN) for supporting the evaluation function by predicting the response of the controlled system has also been implemented. The execution time has been compared with the time achieved by a 1 GHz Intel Celeron PLC. The results have demonstrated that the overall performance of the PLC is better, as it benefits from higher processing capacity and other advantages such as cache and faster memory.

The study presented by Mininno et al. [8] has shown other types of hardware to implement a GA. The board dSPACE 1104, based on a 250 MHz Motorola Power PC microprocessor, has been built for measurement and control applications and is equipped with several AD and DA converters. It can be programmed by Matlab/Simulink. The authors have tested their variation of CGA with the Michalewicz benchmark function and succeeded in using the hardware for online optimization of proportional and integral gains of a discrete PI controller with anti-windup applied to a nonlinear plant.

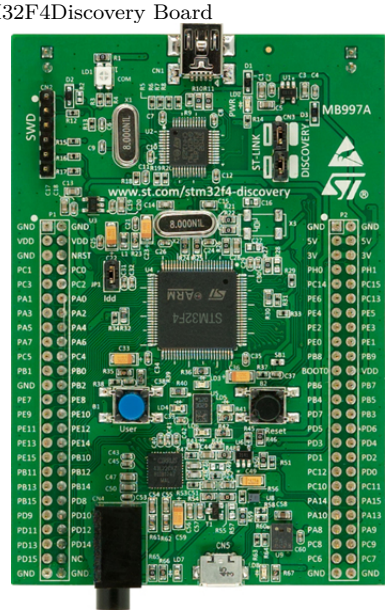
Most existing implementations of GA discussed are related to single objective problems. Taking this into account, the main contribution of this paper is to combine elements presented in the related works to implement a multi-objective genetic algorithm (MOGA) on a simple microcontroller, like the PIC used by Khan et al. [6], instead of the real-time processor used by Dendaluce et al. [7]. A graphical representation of the population performance is also shown, by sending data through a serial port to a computer. The proposed MOGA is evaluated using test functions presented in the literature to validate the distribution of the population through the Pareto frontier. Moreover, the time to execute one generation is compared to the same code running on a computer.

III. Materials and Methods

A. Hardware

The MOGA has been implemented using an STM32F407 microcontroller, mounted on an STM32F4Discovery board, as shown in Figure 2. The microcontroller has a 168MHz 32bit ARM Cortex-M4f CPU with 1Mbyte of Flash and 192KBytes of RAM. It was chosen due to its lower cost and its broad adoption as seen many examples and codes available in the literature. The Cortex-M4f has a single precision floating point unit (FPU) that can execute several mathematical operations on a single CPU cycle.

Fig. 2. STM32F4Discovery Board



The STM32F4 board provides a True Random Number Generator (TRNG) peripheral that can be used for generating a new population and performing selection, crossover, and mutation. The TRNG can speed up the execution of the MOGA because it is different from the usual Pseudo-Random Number Generator (PRNG) used on microprocessors that need to use mathematical functions to calculate every pseudo-random number.

The firmware has been developed using the C language, on IAR Workbench for an ARM CPU. To reduce the overhead, no Real-Time Operational System (RTOS) has been used. The time to execute a generation is measured in CPU cycles to be invariant, assuming different and variable CPU clocks (keeping the same architecture and optimization levels).

B. MOGA Implementation

The MOGA considered in this paper is based on the MOGA presented by Murata and Ishibuchi [15]. This algorithm separates the best individuals of each generation from an external population. In a MOGA implementation, within few generations (lesser than 10), all individuals are in the Pareto Frontier ($f = 1$, see Eq.1) and the algorithm is challenged to spread its individuals throughout the

Pareto Frontier so that they are equally spaced. It can be achieved by using Eq.2 as the performance function. As presented in the study made by Soares et al. [16], where p is the performance of the individual, f is the frontier number, i is the number of solutions that dominate the individual and n is the number of neighbors of an individual, calculated by the algebraic distance of one individual to another, within a chosen radius.

$$f = 1 + 1 \times i \tag{1}$$

$$p = \frac{1}{f} + \frac{1}{n} \tag{2}$$

The steps of MOGA by Murata and Ishibuchi [15] are shown in Tab.II. The frontier number (steps 4 and 14) and the number of neighbors must be found (steps 4/14 and 5/15, respectively) to calculate the performance. When a new generation begins, the current population is preserved, and the selection method is applied to a new population with the same number of individuals is created. Crossover and mutation are applied to this new population. Steps 13 to 15 are performed, and finally, the elitist strategy is applied to reduce the two populations into a single one, which presents the best candidates for each one.

This strategy is applied because selection, crossover, and mutation could result either in better or worse individuals, so the algorithm chooses only the ones that have the best performance. Most of the time, the best individuals are selected by its neighbor number (since the frontier number is 1 for all), resulting in an equally distributed population.

Since each processor architecture has their singularity, some aspects of the MOGA differ for each one. Unlike microprocessors, microcontrollers usually pack small amounts of RAM, which may limit population and chromosome size. Furthermore, most microcontrollers implement a small number of instructions, requiring more clock cycles to run the same operation. Despite all this, a clear advantage of the microcontroller is the fact that it does not need an operating system. A multi-task environment like the one provided by an operating system or an RTOS would require the code to pause after each interaction to briefly release the CPU to perform another task.

The development of a MOGA may change depending on the architecture used. Microcontrollers have little RAM, the variable type used on chromosomes and the fitness value for each limit the size of the population. Microprocessors, however, have a high amount of RAM and can have higher populations. Microprocessors also use RTOS, so the MOGA must be implemented in a way that priority tasks may be executed, for instance by pausing the execution after each generation. This is not a problem for microcontrollers, which only have one specific task.

IV. Results

This section aims to present the results for the execution of five benchmark functions running at the STM32F407 microcontroller. The source code used to create the binary for the microcontroller was ported to run on an Intel Core i5-2520M CPU at 2.50 GHz to compare the demanded CPU cycles for each benchmark function. Since the C Language can be compiled for both platforms, not many changes were necessary for the code to be successfully ported. The following benchmark functions are commonly used in the literature to evaluate the precision, robustness and general performance of optimization algorithms [17]–[19]. As explained on Section III, STM32F407 used its TRNG to generate random numbers to perform selection, crossover, and mutation. The Intel microprocessor, however, used the random function present in the standard C library, with the current time seeding the PRNG algorithm.

TABLE II. MOGA Implemented on the Microcontroller

inputs(*parameters*)
outputs(*population*)

1 initialize t with 0
2 initialize random *population* with *parameters*
3 compute the performance of the initial *population*
4 compute each *individual* frontier
5 compute number of neighbors of each *individual*
6 if generation limit reached, jump to 18
7 store current population
8 increment t
9 apply *selection* method
10 apply *crossover* method
11 apply *mutation* method
12 update *external population*
13 compute the performance of the new *population*
14 compute the *individual* frontier of both *populations*
15 compute number of neighbors of each *individual*
16 apply *elitist strategy* on both *populations*
17 jump to 6
18 return current *population*

- Schaffer Function N. 1, Equation 3
- Schaffer Function N. 2, Equation 4
- Fonseca and Fleming Function, Equation 5
- Kursawe Function, Equation 6
- Zitzler–Deb–Thiele's Function N. 3, Equation 7

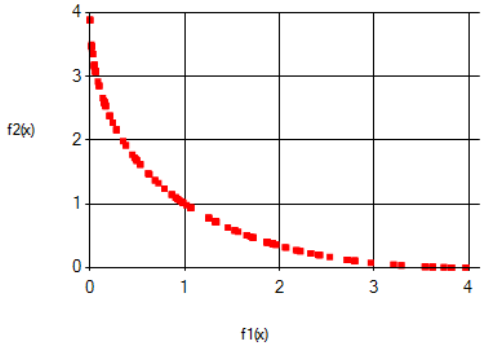
All tests were performed with the following parameters:

- Population size = 100
- Selection Method = Roulette
- Crossover Probability = 0.6
- Mutation Probability = 0.05
- Neighborhood Size = 0.02
- Number of Generations = 200

The first test was done using the benchmark functions described in Eq.3 (Schaffer Function N. 1), where $-A \leq x \leq A$ with $A = 100$. The complexity of this problem increases significantly with A values higher than 10^5. Figure 3 shows that the MOGA was able to find the Pareto Frontier presented in a parable.

$$\begin{cases} f_1(x) = x^2 \\ f_2(x) = (x-2)^2 \end{cases} \tag{3}$$

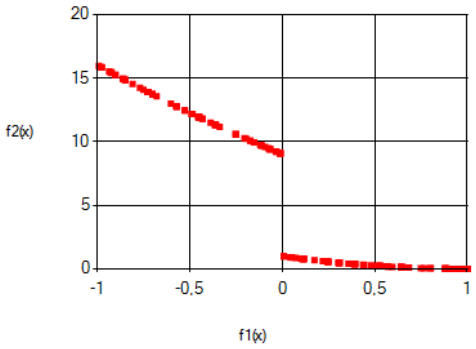
Fig. 3. Schaffer Function N. 1 Results



The second test used the benchmark functions of Eq.4 (Schaffer Function N. 2), where $-5 \leq x \leq 10$. This problem has a discontinuous Pareto Frontier, and it was correctly found by the algorithm, as can be seen in Figure 4.

$$\begin{cases} f_1(x) = \begin{cases} -x, & if x \leq 1 \\ x-2, & if 1 < x \leq 3 \\ 4-x, & if 3 < x \leq 4 \\ x-4, & if x > 4 \end{cases} \\ f_2(x) = (x-5)^2 \end{cases} \tag{4}$$

Fig. 4. Schaffer function N. 2 Results

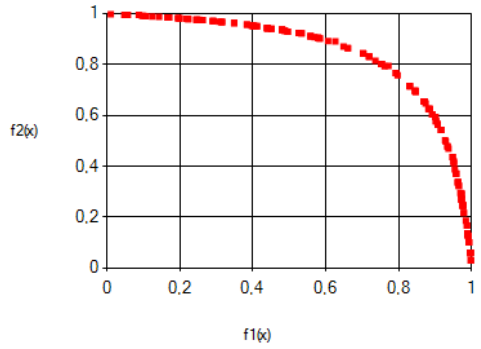


To increase the difficulty of the experiment, Eq.5 (Fonseca and Fleming Function) was tested, where $-4 \leq x_i \leq 4$ and $n = 2$. These functions have two dimensions, which increase the time to execute a generation.

$$\begin{cases} f_1(x) = 1 - exp(-\sum_{i=1}^{n}(x_i - \frac{1}{\sqrt{n}})^2) \\ f_2(x) = 1 - exp(-\sum_{i=1}^{n}(x_i + \frac{1}{\sqrt{n}})^2) \end{cases} \tag{5}$$

Combining the characteristics of the previous benchmark functions, Eq.6 (Kursawe Function) has multidimensional functions with a discontinuous Pareto Frontier, where $-5 \leq x_i \leq 5$ and $1 \leq i \leq 3$. The results shows that the algorithm was able to spread the individuals

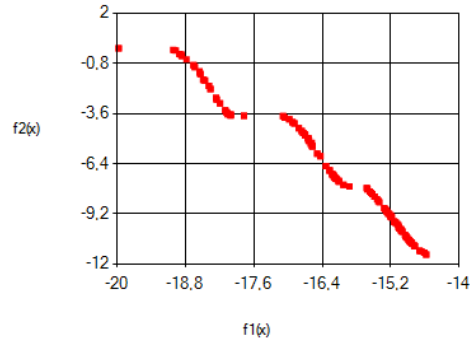
Fig. 5. Fonseca and Fleming Function Results



through the Pareto Frontier, including the spot at (-20,0), as presented on Fig.6.

$$\begin{cases} f_1(x) = \sum_{i=1}^{2}(-10exp(-0.2\sqrt{x_i^2 + x_{i+1}^2})) \\ f_2(x) = \sum_{i=1}^{3}(|x_i|^{0.8} + 5\sin(x_i^3)) \end{cases} \tag{6}$$

Fig. 6. Kursawe Function Results

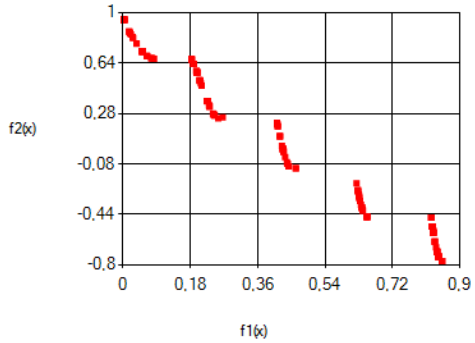


The final test case uses the most difficult benchmark functions and it is presented in Eq.7 (Zitzler–Deb–Thiele's Function N. 3), where $0 \leq x_i \leq 1$ and $1 \leq i \leq 30$. The algorithm should be able to find all five discontinuous curves while computing 30-dimension functions. On Fig.7, it is possible to see that all curves were obtained, but the fourth one (from left to right) has a smaller size than the others. With more generations, the curve with smaller size changed, but all of them were still present on the Pareto Frontier.

$$\begin{cases} f_1(x) & = x_1 \\ f_2(x) & = g(x)h(f_1(x), g(x)) \\ g(x) & = 1 + \frac{9}{29}\sum_{i=2}^{30} x_i \\ h(f_1(x), g(x)) & = 1 - \sqrt{\frac{f_1(x)}{g(x)}} - (\frac{f_1(x)}{g(x)})\sin(10\pi f_1(x)) \end{cases} \tag{7}$$

Presumably, the execution on the Intel microprocessor was expected to require fewer CPU cycles. It stems from the fact that its architecture includes more mathematical instructions and more advanced Floating point unit than the microcontroller. As seen in Tab.III and Tab.IV, the

Fig. 7. Zitzler–Deb–Thiele's Function N. 3 Results



Intel microprocessor took 2/3 of the cycles spent on the microcontroller. The time spent depends on the clock of each device if the architecture is maintained.

TABLE III. CPU Cycles comparison between STM32F407 and Core i5-2520M

CPU Cycles Spent	STM32F407	i5-2520M
Schaffer Function N. 1	6001962	4045695
Schaffer Function N. 2	5813159	3967418
Fonseca and Fleming Function	5761142	4039574
Kursawe Function	6444559	4027721
Zitzler–Deb–Thiele's Function N. 3	6875774	4501426

TABLE IV. Time spent comparison between STM32F407 and Core i5-2520M

Time Spent (ms)	STM32F407	i5-2520M
Schaffer Function N. 1	35.73	1.62
Schaffer Function N. 2	34.60	1.59
Fonseca and Fleming Function	34.29	1.62
Kursawe Function	38.36	1.61
Zitzler–Deb–Thiele's Function N. 3	40.93	1.80

TABLE V. Energy consumption comparison between STM32F407 and Core i5-2520M

Energy Consumed (J)	STM32F407	i5-2520M
Schaffer Function N. 1	13.79	56.70
Schaffer Function N. 2	13.36	55.65
Fonseca and Fleming Function	13.24	56.70
Kursawe Function	14.81	56.35
Zitzler–Deb–Thiele's Function N. 3	15.80	63.00

Tab.V shows the energy consumption for the microprocessor and the microcontroller. The energy used was calculated considering that each device uses its full capacity while executing the MOGA, without considering auxiliary circuits needed for each device. The results show that the microcontroller uses around 1/4 the amount of energy of the microprocessor for each execution of the algorithm.

V. Conclusion

The results shown above demonstrate the viability of a microcontroller to execute the MOGA and find the Pareto Frontier for all benchmark functions. The individuals were well distributed throughout the Pareto Frontier by using Eq.2.

As expected, the time to compute a generation in the STM32F407 microcontroller was 23.13 times higher than the i5-2520M microprocessor. Despite the fact that the microprocessor completed the task quicker, the microcontroller has the potential to be used for different tasks where the execution time isn't the primary requirement. Also, beyond the performance, the price and energy consumption of both platforms must be considered, since the energy consumption of the microprocessor is around four times higher than the microcontroller. It should also be noted that the microcontroller can be used as a standalone device, unlike the microprocessor, which makes it a good choice to be used in embedded systems, such as small robots or drones that can't handle a full-featured computer on-board.

Acknowledgments

The authors would like to thank FAPEMIG, CNPq and CAPES for providing financial support for this project.

References

[1] G. L. Soares, *Algoritmos Determinístico e Evolucionário Intervalares para Otimização Robusta Multi-Objetivo.* Phd thesis, Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais, October 2008.

[2] A. Gaspar-Cunha and J. A. Covas, "Resolução de problemas complexos de engenharia utilizando algoritmos evolutivos multiobjetivo," *Congreso Latino-Iberoamericano de Investigación Operativa*, September 2012.

[3] C. Aporntewan and P. Chongstitvatana, "A hardware implementation of the compact genetic algorithm," in *IEEE Congress on Evolutionary Computation*, pp. 624–629, Citeseer, 2001.

[4] J. C. Gallagher, S. Vigraham, and G. Kramer, "A family of compact genetic algorithms for intrinsic evolvable hardware," *Evolutionary Computation, IEEE Transactions on*, vol. 8, no. 2, pp. 111–126, 2004.

[5] I. J. Griffiths, Q. H. Mehdi, and N. E. Gough, "Microcontroller based genetic algorithms," *Microprocessors and Microsystems*, vol. 21, no. 6, pp. 403–413, 1998.

[6] S. Khan, S. F. Abdulazeez, L. W. Adetunji, A. Alam, M. J. E. Salami, S. A. Hameed, A. H. Abdalla, and M. R. Islam, "Design and implementation of an optimal fuzzy logic controller using genetic algorithm," *Journal of computer science*, vol. 4, no. 10, p. 799, 2008.

[7] M. Dendaluce, J. J. Valera, V. Gómez-Garay, E. Irigoyen, and E. Larzabal, "Microcontroller implementation of a multi objective genetic algorithm for real-time intelligent control," in *International Joint Conference SOCO'13-CISIS'13-ICEUTE'13*, pp. 71–80, Springer, 2014.

[8] E. Mininno, F. Cupertino, and D. Naso, "Real-valued compact genetic algorithms for embedded microcontroller optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 12, no. 2, pp. 203–219, 2008.

[9] J. J. Moreno, G. Ortega, E. Filatovas, J. A. Martínez, and E. M. Garzón, "Using low-power platforms for evolutionary multi-objective optimization algorithms," *The Journal of Supercomputing*, vol. 73, pp. 302–315, Jan 2017.

[10] A. Nasrollahzadeh, G. Karimian, and A. Mehrafsa, "Implementation of neuro-fuzzy system with modified high performance genetic algorithm on embedded systems," *Applied Soft Computing*, vol. 60, no. Supplement C, pp. 602 – 612, 2017.

[11] H. Choudhary, "Difference between microprocessor and microcontroller." http://www.engineersgarage.com/tutorials/difference-between-microprocessor-and-microcontroller. Accessed on 18-November-2015.

[12] R. Mary, "Microcontrollers." http://www.engineersgarage.com/microcontroller. Accessed on 03-November-2015.

[13] J. R. Cárdenas Valdez, E. Z-Flores, J. C. Núñez Pérez, and L. Trujillo, *Local Search Approach to Genetic Programming for RF-PAs Modeling Implemented in FPGA*, pp. 67–88. Cham: Springer International Publishing, 2017.

[14] ARM, "Cortex-r series." http://www.arm.com/products/processors/cortex-r. Accessed on 12-January-2016.

[15] T. Murata and H. Ishibuchi, "Moga: Multi-objective genetic algorithms," in *Evolutionary Computation, 1995., IEEE International Conference on*, vol. 1, p. 289, IEEE, 1995.

[16] G. L. Soares, F. G. Guimarães, C. A. Maia, J. A. Vasconcelos, and L. Jaulin, "Interval robust multi-objective evolutionary algorithm," in *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pp. 1637–1643, IEEE, 2009.

[17] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach," *evolutionary computation, IEEE transactions on*, vol. 3, no. 4, pp. 257–271, 1999.

[18] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," *Evolutionary computation*, vol. 8, no. 2, pp. 173–195, 2000.

[19] S. Huband, P. Hingston, L. Barone, and L. While, "A review of multiobjective test problems and a scalable test problem toolkit," *Evolutionary Computation, IEEE Transactions on*, vol. 10, no. 5, pp. 477–506, 2006.