Genet Program Evolvable Mach (2011) 12:403–427
DOI 10.1007/s10710-011-9137-2

Accelerating floating-point fitness functions in evolutionary algorithms: a FPGA-CPU-GPU performance comparison

Juan A. Gomez-Pulido · Miguel A. Vega-Rodriguez · Juan M. Sanchez-Perez · Silvio Priem-Mendes · Vitor Carreira

Received: 11 February 2011 / Revised: 25 March 2011 / Published online: 20 April 2011

Abstract Many large combinatorial optimization problems tackled with evolutionary algorithms often require very high computational times, usually due to the fitness evaluation. This fact forces programmers to use clusters of computers, a computational solution very useful for running applications of intensive calculus but having a high acquisition price and operation cost, mainly due to the Central Processing Unit (CPU) power consumption and refrigeration devices. A low-cost and high-performance alternative comes from reconfigurable computing, a hardware technology based on Field Programmable Gate Array devices (FPGAs). The main objective of the work presented in this paper is to compare implementations on FPGAs and CPUs of different fitness functions in evolutionary algorithms in order to study the performance of the floating-point arithmetic in FPGAs and CPUs that is often present in the optimization problems tackled by these algorithms. We have taken advantage of the parallelism at chip-level of FPGAs pursuing the acceleration of the fitness functions (and consequently, of the evolutionary algorithms) and showing the parallel scalability to reach low cost, low power and high performance

J. A. Gomez-Pulido (✉) · M. A. Vega-Rodriguez · J. M. Sanchez-Perez
Department of Technologies of Computers and Communications, University of Extremadura, 10003 Caceres, Spain
e-mail: jangomez@unex.es

M. A. Vega-Rodriguez
e-mail: mavega@unex.es

J. M. Sanchez-Perez
e-mail: sanperez@unex.es

S. Priem-Mendes · V. Carreira
Department of Computer Science, School of Technology and Management, Polytechnic Institute of Leiria, 2410 Leiria, Portugal
e-mail: smendes@estg.ipleiria.pt

V. Carreira
e-mail: vmc@estg.ipleiria.pt

computational solutions based on FPGA. Finally, the recent popularity of GPUs as computational units has moved us to introduce these devices in our performance comparisons. We analyze performance in terms of computation times and economic cost.

Keywords FPGA · Evolutionary algorithms · Fitness · Reconfigurable circuits · GPU · Floating-Point · Performance · Parallelism

1 Introduction

The reconfiguration of hardware at logic-circuit levels to suit the application at hand has created a computing area that blurs traditional frontiers between software and hardware. This area, named Reconfigurable Computing (RC) [1], is based on the use of programmable logic devices, mainly Field Programmable Gate Arrays (FPGAs) [2] incorporated in board-level systems. FPGA devices have the benefits of hardware speed and software flexibility, and hence are a good option for many real scientific and engineering applications. Currently, High-Performance Reconfigurable Computing (HPRC) is a promising paradigm [3], in which the possibilities that FPGAs offer are explored. HPRC exploits FPGA capabilities such as high-performance computational units for real applications [4].

FPGA-based applications offer computational solutions based on a number of characteristics: the technologies of modern programmable logic devices (circuits of 65 nm capable of reaching high operation frequencies), the capacity to execute real parallelism of the computing tasks (knowing that, the more parallelism, the more performance) and efficient designs supported by ever-better synthesis tools. Parallelism emerges as the most powerful of these characteristics. The designer should take into account the three possible levels of parallelism in order to exploit to the maximum this advantage (Fig. 1):

(1) A specific-purpose processor designed for a determined problem is efficient when the tasks have a high degree of parallelism. From here on, we refer to this processor as a Hardware Processor (HP).
(2) Some of these processors can compute in parallel within the same FPGA device, depending on the area occupied.

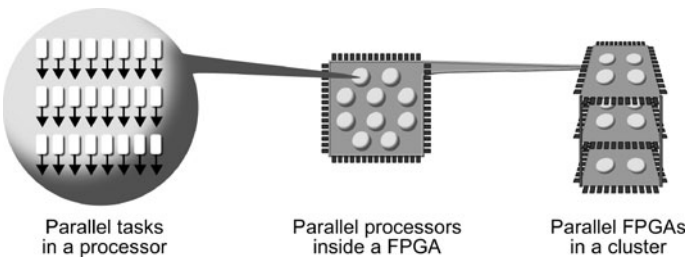

Fig. 1 The three levels of parallelism in HPRC

(3) Some of these FPGA devices can run in parallel by means of a cluster system, which can be driven by a typical master-slave scheme.

The aim of our work is to accelerate the resolution of determined combinatorial optimization problems by means of HPRC. Traditionally, the resolution of large combinatorial optimization problems is done by means of Evolutionary Algorithms (EAs) running on general-purpose CPUs (we refer to a CPU or a computer without distinction from here on). EAs start from an initial population of individuals constituting an instance of the problem to be solved. New individuals are generated by applying a set of evolutionary operators over the individuals of the population, thus determining a new population (a generation). The process is then repeated until a defined stop criterion is reached. There are many bio-inspired algorithms, heuristics and meta-heuristics with a common property: the evolution over time of a population of individuals whose characteristics evolve towards the solution of the problem [5–7]. The process that usually requires the most computation time in the evolutionary algorithm is the fitness evaluation, which depends on the so-called cost or fitness function (F). The fitness function states the individual's degree of optimality to the best one that is sought, and often requires a large amount of floating point arithmetic. The more complex this function and the greater the size of the population, the greater is the computational cost of the EA. Since a fitness function for a given problem has the same formulation for any EA, we focus our design effort and performance analysis on the hardware implementation of the fitness function in order to accelerate any EA.

We may note that the application of the evolutionary operators and processes on each individual of the population is essentially a parallel task; for example, the fitness of one individual can be evaluated at the same time another individual is being evaluated, and likewise the rest of the individuals in the population (Fig. 2). This is only possible in practice if the computational resources permit parallel executions, which are possible with clusters of computers or parallel HPs implemented on FPGAs, but not in mono-processor computers. Thus, the feasibility of a HPRC proposal to integrate FPGAs with EAs in order to solve combinatorial optimization problems comes from the ease with which these problems can be highly parallelized, the high computational consumption required, and the ability of FPGAs to execute real parallelism. Since the biggest resource consumption typically comes from the fitness evaluation, the main goal of a FPGA implementation of an arithmetical processor is to accelerate the computation of F by introducing the largest possible degree of parallelism. This way, we have bet to bet to parallelize both the fitness function and the fitness evaluation of the individuals (fine grain parallelism) instead of parallelizing the EA (coarse grain parallelism), because the area of the FPGA permits more parallel hardware units, and consequently more performance. Then, in a second stage we can leave the software to parallelize the EA in, for example, distributed servers.

When using a single CPU, the fitness evaluation of the population requires a sequential computation of all the individuals, thus multiplying the time consumption of the fitness function by the number of individuals in the population. Although the technology of the current general-purpose processors often offers better

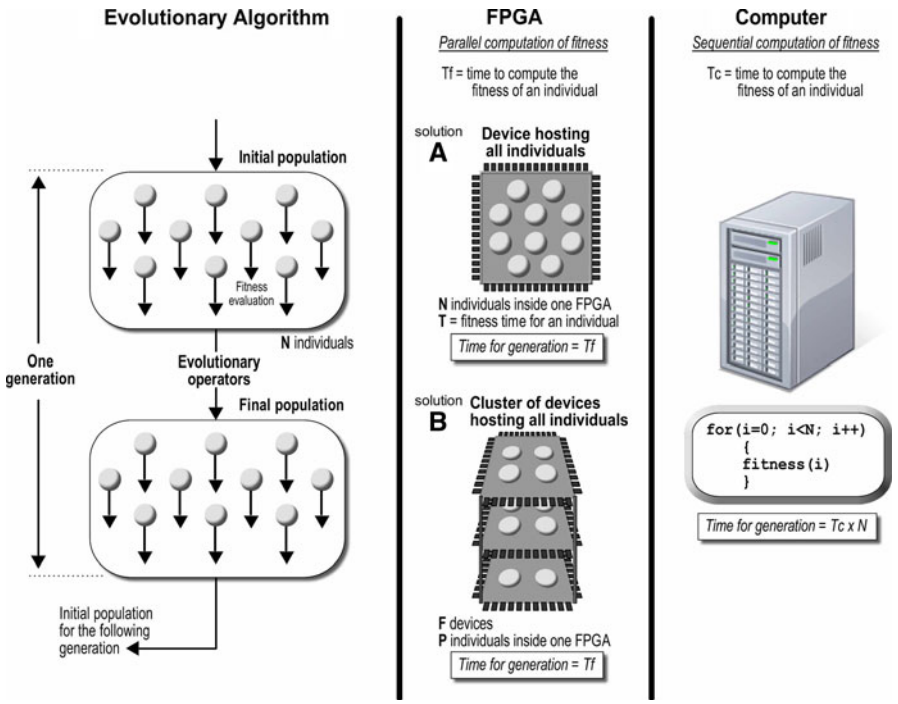

Fig. 2 Parallelism in EAs and FPGAs. When comparing the computation time of solutions based in FPGA parallel processors and in software running on general mono-processor computers, it should be noted that the fitness evaluation of a population is computed in parallel in the FPGA but sequentially for each individual in the CPU

performance than FPGAs for floating-point arithmetic operations, the massive sequential execution cancels this advantage. Hence the viability of a computational solution based on FPGA comes from the ability to process the individuals in a parallel way. If the fitness of an individual in the population is calculated by means of a specifically-designed floating-point arithmetic processor implemented on a FPGA, the goal is to implement a system able to host as many HPs as there are individuals in the population, so that the fitness of the whole population can be evaluated in parallel. This hardware system can be built with only one FPGA (usually mounted on a prototyping board) or with several FPGAs operating in a cluster, depending on the area occupied by one HP in the FPGA, which depends on the particular optimization problem and on the device considered.

It is a well-known issue that it is expected that the fitness evaluation time is the most time consuming part of the EA [8–11]. Speed considerations made some authors even avoid using floating point in time-critical program parts [8]. Other important consideration is the type of applications where FPGA acts as coprocessor. Thus, the complexity and configurability of EAs allows for a rapid processing of the time consuming parts in hardware and leaving other parts to more easily modifiable software. These are the reasons why we only have considered the comparison of fitness times between FPGA and CPU in this paper.

We propose in this work three combinatorial optimization problems suitable for doing a performance analysis of FPGAs versus CPUs. In each case the fitness evaluation involves floating-point arithmetic.

2 Related work

In this section we offer some observations on FPGA versus CPU performance, whether in evolutionary applications or otherwise.

When we consider the implementation of the fitness function in reconfigurable hardware as part of an evolutionary computation framework, we are considering intrinsic evolution. Wang and Lee [12] recognize two approaches to this fact. The first employs FPGAs to compute the fitness of each individual, leaving the computer (or embedded processor) to drive the EA. Following this idea, Glette and Torresen [8] show a built-in PowerPC processor running an evolutionary algorithm on a Xilinx Virtex-II Pro FPGA device. This solution offers hardware acceleration for the fitness function (the more time-consuming part) and flexible evolutionary software running in parallel, using twice as much time as a PC running at 10 times faster clock speed. The second approach considers the FPGA implementation of the whole EA, including the fitness evaluation. For example, Vega et al. [13] parallelize tasks in order to accelerate a Genetic Algorithm implemented on a FPGA for the traveling salesman problem, obtaining better results than software when the problem size increases or when better solutions must be found. Another example is offered by Allaire et al. [14] where a FPGA implementation of a Genetic Algorithm saves the real-time requirements for the path re-planning of unmanned aerial vehicles, a very slow task which is accelerated thanks to the parallelism achieved. This second approach is more complex and highly problem-dependant, due mainly to the limited FPGA area. For this reason, we consider in this paper three large combinatorial optimization problems that lend themselves to the first strategy. In addition, many other authors (Koza et al. [15] and Qiu et al. [16] are representative works) show results that reinforce our interest in accelerating the fitness evaluation by means of FPGAs.

On the other hand, the application of reconfigurable computing to evolutionary computation is a case that can help to show the advantages of FPGAs in accelerating CPU-based applications, although any highly-parallelizable problem involving floating-point calculations would be well suited. At a higher level, for many scientific applications (not only EAs), the FPGA versus CPU competition has been studied from different points of view. Within this field, our interest is focused on floating-point arithmetic, because of the fitness functions in many optimization problems. Thus, and as a representative case, Underwood and Hemmert [17] examine the floating-point performance for some basic linear algebra subroutine functions computed on microprocessors and FPGAs. Their analysis concludes that FPGAs outperform current CPUs on memory bandwidth sensitive double precision floating-point operations. Although FPGAs are limited by a peak floating-point rate, the trends indicate that the performance gap will widen, so FPGAs offer great

promise for scientific computing. In other case, Sukhwani et al. [18] describe three floating point intensive applications where FPGAs are highly competitive.

The computation of EAs can be tackled by means of FPGA-based clusters in order to achieve lower computing times, as in Pedraza et al. [19]. Therefore, FPGA-cluster emerges as a low-power and low-cost promising platform to solve large problems. Bioinformatics [20], signal processing [21] and streaming algorithms [22] are some examples of FPGA-based clusters applications.

Finally, since the development time is usually higher for FPGA designs than for CPU programming, this gap is currently being reduced because of the appearing in recent years of new high-level hardware description languages different to VHDL (Handel-C, System-C, Mitrion-C, etc.) that facilitate to the HPRC an easier and quicker way to program algorithms abstracting away the hardware as far as is possible [23]. In addition, if we want to take advantage of the parallelism in current dual or quad-cores CPU chips or in multiprocessor clusters, we must to apply programming techniques more complex (threads, message passing, etc.), extending the development time for the final application.

3 Problem formulation

As indicated in the introduction, we present three different combinatorial optimization problems with a common characteristic: the fitness evaluation involves floating-point arithmetic though with different degrees of parallelism.

3.1 The error correcting codes optimization problem

The repetition of a corrupt message is an obstruction in real-time communications, especially in multi-point systems, such as phone, television and digital radio. In these cases, the information should be delivered with low delay, for which the use of techniques avoiding transmission overloads are adequate. Since during the transmission of digital information through a channel, practically inevitable errors are produced (Fig. 3), Error Correcting Codes (ECCs) as defined by Hamming [24] could be used to recognize and correct them.

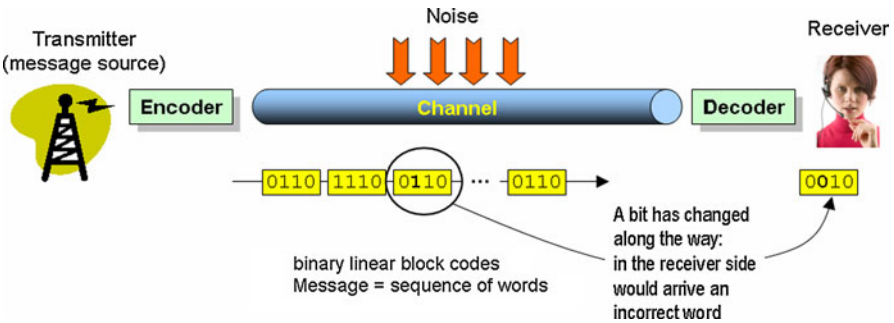

Fig. 3 Errors in a basic communication system

Many types of ECC exist. In this work the binary linear block codes are used. In this type of code, an ECC is a set of codewords of n bits used to transmit a message. The size of a code (M) is the number of different codewords in the code. When a message is transmitted, its binary string is sent through the physical environment building a flow of bits that can suffer modifications while crossing it. An error in a codeword is produced when a bit has changed along the way, such that on the receiving side an incorrect codeword arrives. The distance ($d_{\min}$) from a code is the minimum Hamming distance between pairs of codewords of the code, where the Hamming distance d between two codewords x and y is the number of positions in which they differ (the minimum number of replacement operations to convert x into y). An optimum code is that one that maximizes d. To find this code is a very hard task. For example, for a formulation of the problem of size $M = 24$ and $n = 12$, approximately 10^{63} different codes could be generated. Therefore, an exhaustive search is clearly out of the question, making EAs a good choice. Some EAs have been applied to this problem, such as GAs [25], Simulated Annealing [26] and Parallel Hybrid Heuristics [27].

As a fitness function to maximize, one could use $d_{\min}$, but this measure is not very useful, because it offers a low level of information about the internal advantages of two competing solutions. A more precise fitness function [25] is shown in (1), where x is the code and d_{ij} is the Hamming distance between codewords i and j.

$$f(x) = \frac{1}{\sum_{i=1}^{M} \sum_{\substack{j=1 \\ j \neq i \\ d_{ij} \neq 0}}^{M} \frac{1}{d_{ij}^2}} + \left(\frac{d_{\min}}{12} - \frac{d_{\min}^2}{4} + \frac{d_{\min}^3}{6} \right). \tag{1}$$

3.2 The radio network design optimization problem

The Radio Network Design Problem (RND) originates in the context of wireless communication technologies. An efficient design for a radio transmitting network is a relevant issue due to the continual increase in the user population of the radio-communications-associated services which demand more efficient coverage in wide geographic areas. In short, the RND problem consists of minimizing the number of transmitting base stations (referred to from here on as antennas) and establishing their optimal locations, with the goal of obtaining the maximum coverage area and providing services to the largest number of terminals.

An antenna transmits a radio signal according to its type of coverage. In this work we consider propagation models of simplified waves such as the omni-directional and square, with variable radius. In addition we have defined a digital model of the ground, in which the area is divided into sectors and locations that act as nuclear units of information. Thus, the area consists of a rectangular network, where each coordinate (x, y) represents a possible antenna location. The assumption is that only a well-defined number of valid locations to place antennas exist. For example,

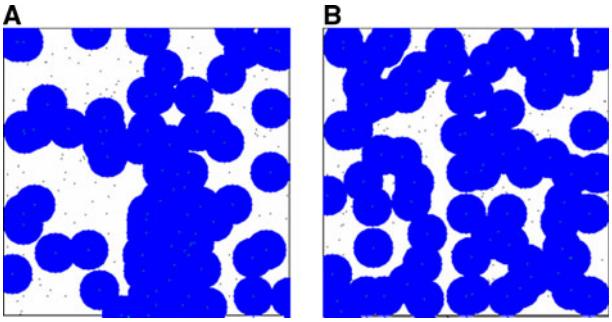

Fig. 4 **a** 61% coverage, **b** 74% coverage

Fig. 4 presents a simple example of the problem, wherein we search for a set of antennas which can reach the maximum coverage area in a terrain of 257 × 257 points, with 349 predefined valid locations for antennas with omni-directional coverage of 35-point radius. Depending on the choice of the antenna locations, the resulting coverage can be completely different.

If some antennas are close enough to one another, their coverage areas overlap, so the locations inside these areas can have different degrees of coverage. For this reason, the information stored in each position of the network must reflect the following data: degree of coverage; whether it is a predefined position available for the placement of an antenna; whether or not an antenna is currently placed there; antenna propagation type (square or omni-directional). The fitness function can be used to measure the quality of a set of antennas placed in any given manner in the network. This function can be obtained from the coverage rate and the number of antennas [28, 29], as shown in (2).

$$F = \frac{\text{Coverage}^2}{\text{Antennas}}. \tag{2}$$

In a real study of this problem we have to determine in the first place the set of available localizations for the antennas, excluding those where they cannot be placed (public areas, certain roofs, rivers, etc.). Afterwards the goal is to achieve the maximum level of coverage for the smallest number of antennas. This is a typical NP-Hard optimization problem for which some EAs have been tested [30, 31] with success.

We considered two additional terrain maps to increase the computational efforts. The first is a real map, based on the city of Málaga, in Spain (Fig. 5), where 1,000 predefined localizations were determined for the placing of 100 antennas. The territory, of 4,25 km × 6,4 km, was codified into a network of 300 × 450 cells (135,000 points in the grid), where each one represents a terrain of approximately 15 m × 15 m. Taking into account the presence of the sea, mountains, public areas and other prohibited zones, the maximum possible coverage is 95.52\%. The second concerns a theoretical case where the computational cost is the highest: 724 × 724 grid points, with 2,000 predefined available localizations for placing 300 antennas.

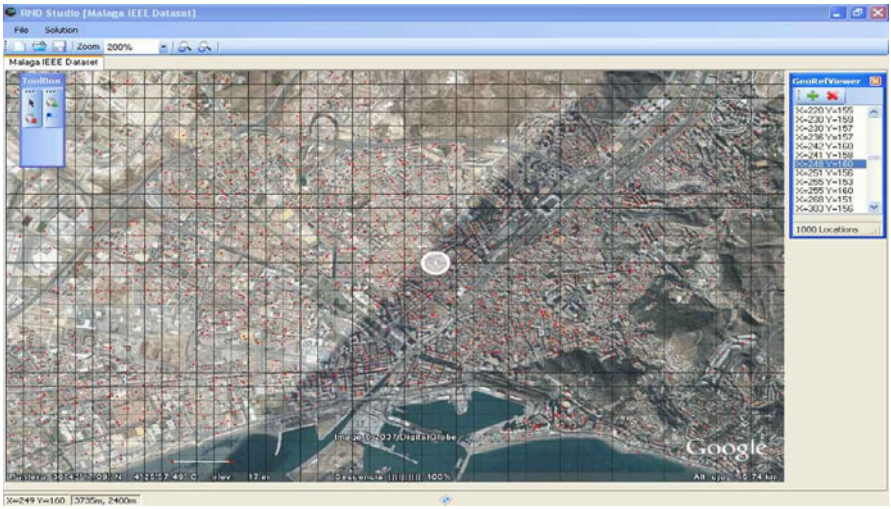

Fig. 5 Map of high computational cost, codified with a 300 × 450 grid where 100 antennas must be placed on a predefined set of 1,000 available locations. The shadowed areas show the maximum possible coverage

3.3 The X-ray diffraction profile adjusting optimization problem

Peaks in X-ray diffraction patterns can be modeled using analytical functions. For example, the widely used pseudo-Voigt function [32] represents, in many cases, a good approximation to the real shape of X-ray diffraction profiles. This is especially important when strong overlap between adjacent reflections arises, and it is necessary to separate the contribution of the different peaks, as for example to obtain information about microstructural quantities from the parameters describing the shape and width of the peak [33–35]. For severely overlapped reflections, many peak combinations (with different parameters) can exist generating the same or similar experimental profile. After obtaining the experimental data (usually in the form of intensity vs. angle), and determining how many peaks there are in a particular region of the spectrum, the goal is to fit a model profile to the data. In this regard, if a profile has two overlapped peaks, many two-peak sets generate model profiles that fit reasonably well to this experimental profile (the XRAY optimization problem). In order to determine the best solution, the fitness function quantitatively assesses the fit to the experimental profile.

We have used as benchmark the simulated step-scanning diffraction profile shown in Fig. 6. This profile has been generated between 25° and 35° with a step of 0.01°, assuming radiation, and with two strong overlapped peaks modeled by means of the pseudo-Voigt function (including some additional components in order to perform a realistic simulation).

The pseudo-Voigt function is defined in (3) where I_0 is the maximum intensity of the peak, x_0 is the position of the maximum, w is the half width at half maximum

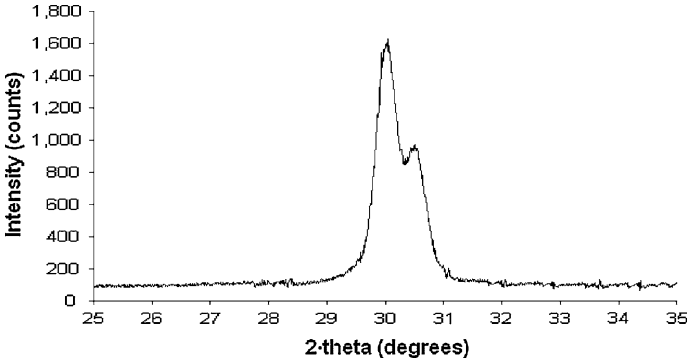

Fig. 6 The simulated diffraction profile with two very strong overlapped peaks used in this work

(HWHM), and η is the partition factor, a number between 0 and 1. The parameters characterizing the two pseudo-Voigt functions describing the peaks included in the profile were the following: $I_0(1) = 1000.0$, $x_0(1) = 30.0^\circ$, $w(1) = 0.2^\circ$, $\eta(1) = 0.5$, $I_0(2) = 500.0$, $x_0(2) = 30.5^\circ$, $w(2) = 0.2^\circ$ and $\eta(2) = 0.5$. Here, the variable x corresponds to the angular variable 2θ, usually expressed in degrees.

$$pV(x) = I_0 \left[\frac{\eta}{1 + \left(\frac{x - x_0}{w}\right)^2} + (1 - \eta)\, e^{-\ln 2 \cdot \left(\frac{x - x_0}{w}\right)^2} \right]. \tag{3}$$

The test profile with two strong overlapped peaks was considered for experimental purposes. The objective of the optimization process is to establish how many peaks there are (although in our case this information is assumed in the generation of the profile) and which are the parameters characterizing the pseudo-Voigt functions (I_0, x_0, w and η) that provide the best fit to the data. In other words, it must be determined which one of the many two-peak sets (possible solutions) represent the best fit to the experimental profile. For this purpose, and according to the usual practice in X-ray diffractometry, we define the chi-squared function, χ^2 as the fitness function (4), where: N is the number of data points (1,001); $y_{i,obs}$ is the value of the intensity measured at the ith step (this intensity corresponds to the simulated profile); $y_{i,cal}$ is the value of the intensity calculated from the model at the ith step (obtained from the model using two pseudo-Voigt function and a linear background); $p_i = 1/y_{i,obs}$ is the weight assigned to each intensity; and $\vec{\alpha}$ is a vector representing the parameters of the problem to be optimized (I_0, x_0, w and η for each peak, and the coefficients of a linear function describing the background). The lower the fitness, the closer the solution is to the objective. Therefore, the effort of the problem consists of finding the vector $\vec{\alpha} = [I_0(1), x_0(1), w(1), \eta(1), I_0(2), x_0(2), w(2), \eta(2)]$ in such a way that F be minimum.

$$\mathrm{F} = \chi^2 = \sum_{i=1}^{N} p_i \left(y_{i,obs} - y_{i,cal}(\vec{\alpha}) \right)^2. \tag{4}$$

4 Design of arithmetic processors in reconfigurable hardware for fitness computation

Several factors must be taken into account in order to design the HP for fitness computation such as the arithmetic operations of the processor, which design methodologies are more suitable and the characteristics of the prototyping platforms. The next subsections explain in a summarized way the following aspects: methodologies, tools, hardware resources (FPGAs, boards, CPUs), design strategies and implementation keys. Details and codes of the designs can be found in [36].

4.1 Design methodologies and tools

The following tools have been used for designing the HP: Xilinx ISE 9.2i, ModelSim 6, VHDL and Handel-C [37]. Several facts have been taken into account in the designs:

1) It has been decided to design custom HPs instead of using embedded processors such as MicroBlaze, because the objective is to have the highest possible speed.
2) The common characteristic in the fitness functions considered is floating-point arithmetic. Thus, the Xilinx Core Generator tool implements the basic arithmetic operators, looking for a balance between using several copies of the same operator (to implement parallel operations) and the area occupied for a determined FPGA device (because the EA performance increases the more HP units can work in parallel in the same FPGA device).
3) It is necessary to take into account the characteristics of the prototyping board used, because this determines different designs for the interfaces with I/O devices.
4) In order to reach the maximum performance, we have attempted to parallelize as far as possible, optimized the synthesis process in speed and used digital clock managers to match the on-board oscillator to the maximum reported clock frequency.

It is important to say that all the HPs were implemented using the synthesis tool with the default options. The occupation of the area (number of FPGA slices occupied) gives us a basis on which to calculate the maximum number of HPs able to work in parallel in the same FPGA device. Also, the maximum reported operation frequency is used to select the proper frequency for the on-board oscillator. After generating and loading the configuration bit stream onto the FPGA, the processor gave the results with an elapsed time measured using the existing displays on the board.

4.2 Hardware resources used: FPGA devices, prototyping boards, CPUs and clusters of computers

The architecture design of the HP is conditioned by the FPGA characteristics and the prototyping board. For this reason we opted for several boards (Digilent

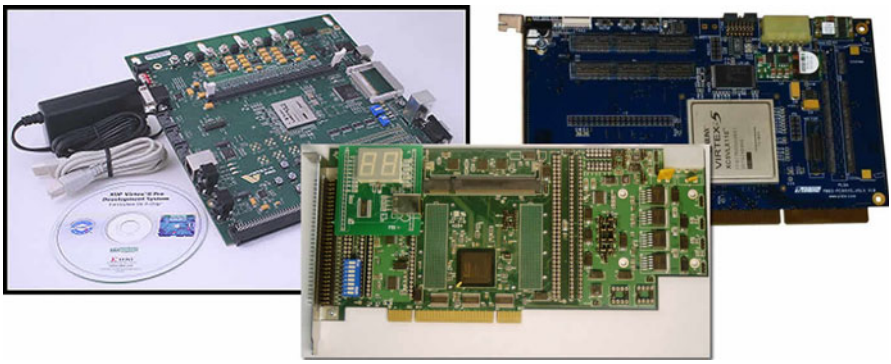

Fig. 7 Prototyping boards used in this work. From the left to the right: Digilent XUPV2P, Enterpoint Broaddown2, and PLDA SYSV5

Table 1 Hardware resources arranged by FPGA versus CPU

Tech	Reconfigurable computing		Computer	
Year	Xilinx FPGA	Board	CPU	Memory
130 nm	Virtex2 Pro	Digilent	Pentium4	1 GB
2002	xc2vp30	XUPV2P	2.4 GHz	RAM
90 nm	Spartan3 2 k	Enterpoint	Pentium4	1.5 GB
2003	xc3s2000	BroadD2	3 GHz	RAM
65 nm	Virtex5 330	PLDA	Core2Duo	4 GB
2006	xc5vlx330	PciXsysV5	E6750 2.6	RAM

XUPV2P, Enterpoint Broaddown2 and PLDA PCIXSYSV5) with FPGAs of different technologies, as shown in Fig. 7. Table 1 shows the FPGA prototyping platforms used as well as the general-purpose CPUs used for comparing the performance results. To establish the FPGA vs CPU comparison, it was taken into account that the devices should be time contemporary and with similar technologies.

When EA is applied to real problems, the parallel execution of the fitness of all the individuals in the population is only possible if the FPGA has enough area to host all the required HPs. Each board shown in Fig. 7 hosts only one FPGA device, where a maximum determined number of parallel HPs can fit. In this case, it would be necessary to implement cluster systems based on FPGAs. For this purpose, we have used a model of these boards (Digilent XUPV2P) to build a FPGA cluster which is easily scalable. The results in this cluster are estimated from the results found in one board, and are compared to those obtained from an available cluster of computers, with the purpose of analyzing the performance with respect to computation time and financial operation cost derived from power consumption. Figure 8 shows both clusters, whose characteristics are listed in Table 2.

Fig. 8 FPGA and CPU clusters used to compare performance

Table 2 Characteristics of the clusters

	Cluster of CPUs (year 2005)	Cluster of FPGAs (year 2006)
Node	Intel Pentium IV Xeon dual-core 64bits 3 GHz	FPGA Xilinx Virtex2 Pro (xc2vp30-7ff896)
Power	95 W	<1 W
Nodes	10	Variable (currently, 12)
Other	1 GB DDR, Gigabit Ethernet, etc.	RS232, Ethernet, USB, 256 MB RAM, 9 V, etc.
Cost	13.200 euros (entire cluster, excluding refrigeration)	1 node: 226 euros. 58 nodes: 13.108 euros

4.3 Design strategies

We have taken into account that there are two possible design strategies to improve the performance: increasing the parallelism degree of the fitness computation and increasing the number of parallel HPs in the FPGA. But both are contradictory: the more parallelism degree in the calculus sequence, the more area occupied by the HP due to the necessity to add repeated floating-point modules. We have chosen to increase the parallelism degree of the fitness computation because the number of HPs could always be increased in larger and newer FPGA and also by adding more devices to a cluster system. Moreover, the performance loss associated to a slower fitness computation would also be transferred to a cluster system.

4.4 ECC processor implementation

Since most of the related studies with FPGAs implement ECC modules, no study that approaches the quality of the ECC solution through a fitness function has been found.

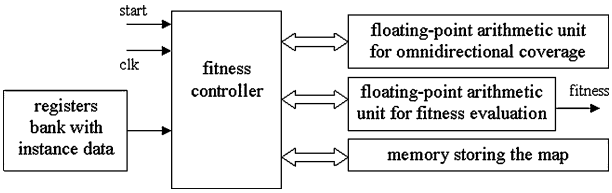

Fig. 9 RND fitness processor diagram

The larger the code, the greater the number of parallel HPs needed, following the quadratic polynomial shown in (5). For example, for $M = 10$, 45 parallel HPs are needed, and for $M = 24$, 276. The HP evaluates all the combinations of codeword pairs. A controller drives the operations of the fitness function (some of them also parallelizable), the floating-point arithmetic and the overall system functionality.

$$\text{Parallel units} = \frac{M(M-1)}{2}. \tag{5}$$

4.5 RND processor implementation

The prototype designed carries out a coverage evaluation for simple configurations of the problem: a 287 × 287-point network; 349 allowed positions (predefined); 49 antennas. This processor allows configuration of the antenna type (square-shaped or omni-directional coverage) and its maximum propagation radius. To select any configuration, it is enough to modify the value of certain registers through board switches. In order to measure the board execution time and determine its efficiency, the processor carries out 1,000 fitness evaluations in a sequential manner. Then the average time of an evaluation is obtained.

Figure 9 presents the architecture of the prototyped processor. The processor uses an on-chip memory where the characteristics of the terrain are stored, so each network point is linked to a 2-byte memory word in an 82,369-address map to represent the 287 × 287 network. The controller is the most important unit in the processor. It was programmed in Handel-C and compiled to VHDL. This controller processes the main operations of the coverage calculation, with the exception of all the floating-point arithmetic operations, which are carried out by other units. In addition, the controller manages the initialization, evolution and termination of the process, the input/output communication and the accomplishment of the memory writes and reads. The mathematical operations for the omni-directional coverage and for the fitness function require floating-point arithmetic, such as addition, multiplication, power and square root. For this purpose we have designed two co-processors that carry out the necessary operations. The results from these floating-point co-processors are sent to the controller, which uses them for the final coverage processing.

4.6 XRAY processor implementation

The scheme of the processor is shown in Fig. 10. This circuit gives the input values of the parameters ($I_0(1) = 1{,}000.0$, $x_0(1) = 30.0$, $w(1) = 0.2$, $\eta(1) = 0.5$, etc.), and

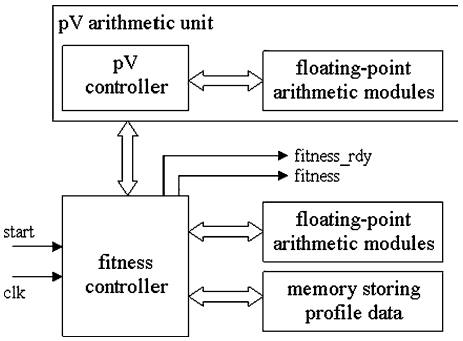

Fig. 10 XRAY fitness processor diagram

reads the output signals (the $fitness_{rdy}$ signal indicates when the fitness computation has been finished and the *fitness* signal indicates the value of the fitness function). The architecture of this HP has the following parts: a memory storing the data of the profile to be processed, a set of floating-point arithmetic units, a control unit and a pseudo-Voigt (pV) arithmetic unit. The memory storing the data of the profile to be processed is a Read-Only-Memory (ROM) device. It stores the samples vs. degrees information of the profile used as benchmark (Fig. 6). The size of this memory is 832 bytes (512 addresses for 13-bit words). Any profile can be loaded into this memory through an initialization file. A set of floating-point arithmetic units are needed to perform the operations described in (4). The controller drives the sequence of the fitness calculation; it needs to read the profile data coming from the memory, and give them, in the appropriate order, to the floating-point arithmetic units, setting accurate control signals. This unit is also connected with the pV arithmetic unit. Finally, the calculated fitness value is read from this control unit.

A design objective that would contribute to an increase in performance is introducing a high degree of parallelism. Theoretically, this is possible in the processor because from (4) the circuit could carry out N floating-point adders in parallel. But, due to the limited size of the FPGA, to introduce N floating-point adders, N memories and N pV arithmetic units is not possible. Therefore, the parallelism advantages of the hardware must come from the processor.

The pV arithmetic unit computes the operations described in (3) according to the input signals that arrive from the test circuit. This unit consists of two parts: a set of floating-point arithmetic units and a pV controller. There is a reason why several repeated units of the same floating-point arithmetic module have been included: the controller can run parallel operations, with an increase in the final efficiency. To duplicate some of these modules has a low occupation cost for the FPGA. The pV controller drives the necessary operations of the pV arithmetic unit. It controls the calculus sequence of the operations in a parallel way, thanks to the duplicated floating-point units. In Fig. 11 the parallel operations are shown. The operations corresponding to the exponential part in (3) (step 5b in Fig. 11) have been computed by means of an iterative algorithm, also with parallel operations. Finally, the calculated value of the fitness is read from the output of the pV controller by the fitness controller.

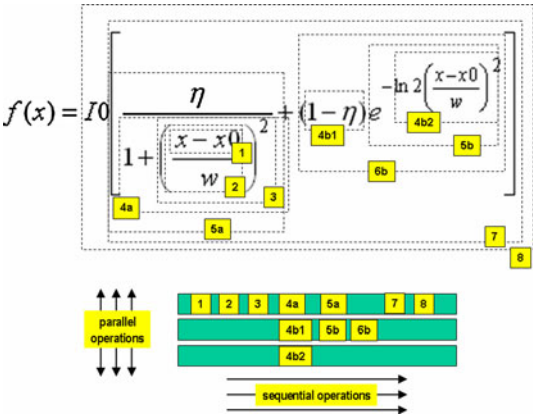

Fig. 11 Parallel operations in the pseudo-Voigt function. The calculus sequence is divided into 8 steps, numbered in yellow boxes. Steps #4 and #5 consist of 3 and 2 parallel operations respectively

5 Performance results

It is important to compare the FPGA results with those obtained from software running on CPUs, in order to analyze the efficiency of the HP as in [17]. Thus, we have developed software (optimized to reach the maximum possible speed) for implementing the same operations as were performed on the FPGA. This software was run on all the CPUs and reconfigurable computing boards listed in Table 1, taking into account the similar technologies and ages of the FPGAs and CPUs in order to make effective comparisons. In this section we present the experimental results and the performance analysis in two parts: firstly we compare one FPGA with its corresponding CPU according to Table 1, and secondly we compare two clusters of FPGAs and computers.

5.1 Comparing one FPGA and one CPU

As mentioned earlier, we take as the measure of performance for this case the computation time. The time measurements have been made by repeating the fitness function computation thousands or millions of times (depending on the problem) and measuring with a stopwatch the time from the activation of a determined switch to when the LED displays show certain information. Here is necessary to show that the FPGA fitness results match up to the ones obtained from the software running on the CPUs.

The most important results to analyze are shown in Table 3, Figs. 12, 13 and 14. Table 3 shows the information related to the synthesis processes of each HP: the operation frequency (selected from the reported maximum frequency after the FPGA routing for a single HP), the area occupied by one HP, and the maximum number of HPs able to fit on the FPGA so as to run in parallel.

The maximum number of HPs is obtained performing a large set of implementations until the occupied area constraint or the memory requirements for the synthesis have been reached. For each implementation, a lightly different frequency is reported. Figure 12 shows the trend to reduce the frequency lightly when the

Table 3 FPGA synthesis results

Devices	Problem	Occupied area for a single HP (%)	Frequency reported for one HP (MHz)	Frequency selected for one HP (MHz)	Max.number of HPs in a single FPGA
Virtex2-Pro	XRAY	43	205	200	2
	ECC	68	105	100	1
	RND	35	43	40	2
Spartan3-2 K	XRAY	29	102	100	3
	ECC	45	61	50	2
	RND	24	27	25	4
Virtex5	XRAY	5	190	175	26
	ECC	8	110	100	26
	RND	3	56	50	15

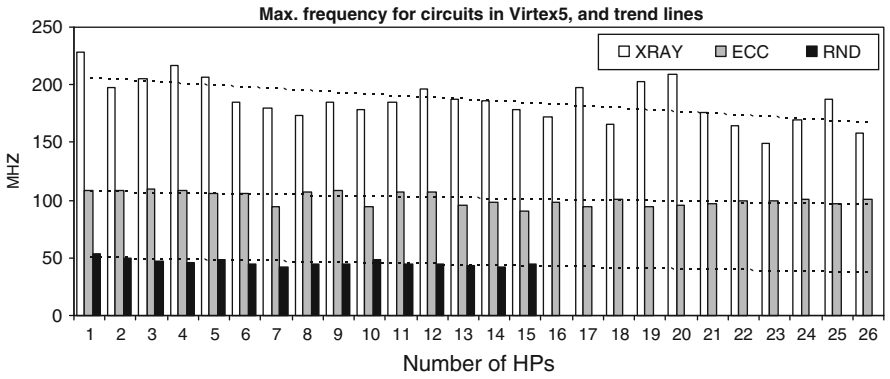

Fig. 12 Maximum frequency reported by the FPGA synthesis for different number of HPs working in parallel in the device, and their trend lines

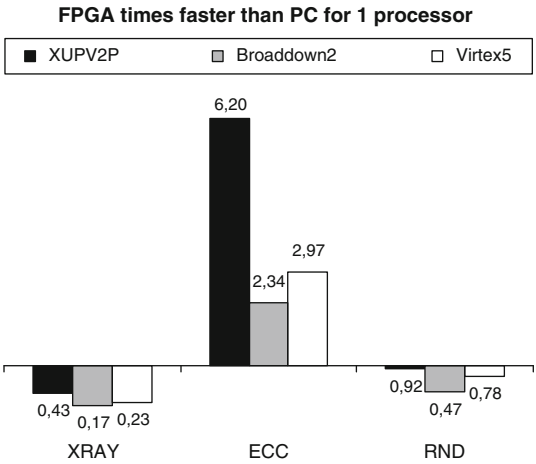

Fig. 13 Computation time improvement of FPGA versus CPU for one HP

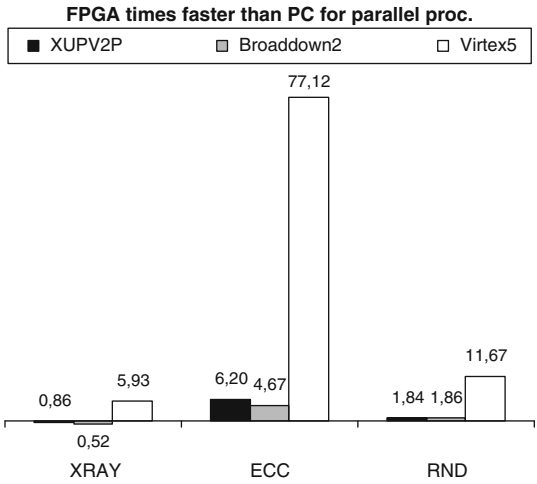

Fig. 14 Computation time improvement of FPGA versus CPU for many HP running in parallel as fitting into one FPGA

number of processors increases. For example, considering the Virtex5 device, the reduce rates for each additional HP in the circuit are 1.49, 0.46, and 0.52% for the problems XRAY, ECC, and RND respectively. Therefore, we can use these rates to extrapolate the calculations in the scalability simulations.

We have selected operation frequencies a little smaller than the ones reported by the synthesis for a single HP, in order to select values easily for the on-board oscillators and to measure the computation times.

The computation time for one HP is the same as for all the parallel HPs that fit on the FPGA, and it is measured by the repeated fitness evaluation. This time is compared with the one obtained in the computer. To measure the processing times in the general-purpose CPU, we have compiled and run C codes exactly reproducing the same operations as used in the FPGA designs, bearing in mind that, in this case, many operations cannot be parallelized and are programmed in a sequential manner. The application measures and displays the elapsed time after executing the same number of fitness evaluations as were performed in the prototyping boards.

The performance comparison between FPGAs and CPUs is evaluated from the computation time (T_{CPU} for the CPU and T_{FPGA} for the FPGA). T_{FPGA} includes the time spent to start and control the computation (load from memories, loop driving, etc.) and to display the results (data conversion, controlling leds and seven-segment displays, etc.). T_{CPU} also includes a small amount of additional time (data preprocessing and postprocessing, active services of the operating system, etc.). The comparison of both times has a real-world interest: considering overall application time had the highest value to the applications science [38]. Thus, we include system dependant FPGA and CPU overhead time when comparing CPU and FPGA based application performance.

One important issue in the comparison methodology was to use devices of the same CMOS process. We think it would be good to make the comparison with devices of similar CMOS process and year of appearance as comparison rule, applying the best hardware resources available every time, although it is not always necessary [39, 40].

We understand as performance the inverted computing time [41]. To compare the performance between FPGAs and CPUs, we say that the FPGA is T_{CPU}/T_{FPGA} times faster than the CPU if $T_{\mathrm{FPGA}} < T_{\mathrm{CPU}}$. According to this definition, Fig. 13 shows how much faster than CPUs the FPGA computes for only one HP. Obviously, the more HPs that are computing in parallel (more individuals of the population being evaluated at the same time), the greater the average performance improvement. In this case, in order to make a realistic comparison, the software version running on the computer executes the fixed number of iterations of the fitness function multiplied by the number of parallel processors in the FPGA device. Thus, Fig. 14 shows the improvement in computing time for all the HPs (running in parallel) that fit in one FPGA, versus the same number of HPs (running sequentially) in one CPU.

Some EAs performed better on FPGAs than others due to two factors: the different parallelism and area occupation degrees in their fitness equations. Thus, we can see that, for all the FPGA devices considered, the results obtained for the ECC problem are very good, with the FPGA performances ranging between 6 and 77 times faster than the corresponding CPU. Its fitness function has the highest parallelism degree of the problems considered, but it occupies more FPGA area. For the RND problem (whose fitness function has the lowest parallelism degree), the Virtex5 FPGA is 12 times faster than its corresponding CPU, due to the lowest area occupation, which generates 15 parallel HPs able to fit on this device. Finally, the XRAY problem offers worse results because of its medium degrees of parallelism and area occupation; nevertheless, considering the Virtex5 device, an acceptable speedup is achieved.

The fitness function is applied to all the individuals of the population in a linear manner (where the coding is equal for all individuals), so the linear scaling from the performance numbers of a single HP repeatedly working on the same input could give us a good approach to real cases.

5.2 Comparing FPGA and CPU clusters

We can improve the FPGA performance by increasing the clock frequency of the circuit. This requires a set of timing closure techniques such as efficient design, synthesis adjustments and implementation constraints. Using these techniques a greater frequency is reached but only by a small percentage. The most effective way to increase significantly the performance of the hardware system is to distribute many parallel fitness processors in more than one FPGA device. We can choose to arrange the FPGA boards in a cluster way as shown in Fig. 8, using a master-slave-based control system, or to use boards with more than one FPGA device. Pursuing this idea, the performance increases in a linear manner. Table 4 shows the results obtained considering two or more devices as nodes of a FPGA-based cluster, where NF is the number of nodes in the cluster. The more FPGA devices are used, the more time improvement is obtained, giving unquestionable gains. If the population size in the EA are surpasses the FPGA-based cluster area capacities, we can always split the population and apply in parallel the fitness function during few sequential loops.

We analyze here the performance of the two clusters described in Table 2. The results are based on the measurements obtained from the different FPGA devices for the optimization problems considered. In this case, in addition to the computation

Table 4 Scalability of the FPGA cluster: Computation time improvement of a NF-nodes FPGA cluster against CPU

Devices	Problem	$NF = 2$	$NF = 4$	$NF = 8$	$NF = 16$	$NF = 32$
Virtex2Pro vs. P4 2.4 GHz	XRAY	1.7	3.4	6.9	13.8	27.5
	ECC	12.4	24.8	49.6	99.2	198.4
	RND	3.7	7.4	14.7	29.4	58.9
Spartan3-2 K vs. P4 3 GHz	XRAY	1.0	2.1	4.1	8.2	16.6
	ECC	9.3	18.7	37.4	74.8	149.5
	RND	3.7	7.4	15.0	29.8	59.6
Virtex5 vs. Core2 2.6 GHz	XRAY	9.1	18.2	36.5	73.0	146.0
	ECC	71.2	142.4	284.8	569.5	1,139.0
	RND	51.3	102.7	205.3	410.7	821.4

time, we include as a performance parameter the economic cost. This cost comprises the operation cost (due to the power consumption of the cluster and the refrigeration system) and the acquisition cost (the price of the clusters). We have taken into account the power consumption of the processors [42] and the electric rates [43] in order to evaluate the economic cost.

We consider two cases in order to analyze the performance: clusters with the same acquisition cost and clusters giving the same computation time.

5.2.1 Clusters with the same acquisition cost

To assume the acquisition cost of both clusters is the same, we have to add 58 nodes (boards) to the FPGA cluster, as we can see in Table 5. In this case, the performance of the FPGA cluster is very high with regard to the cluster of CPUs, whether in computation time or in power consumption. We emphasize two results:

(1) The FPGA is 4 times faster in the worst case (the XRAY problem) and 29 times faster in the best case (the ECC problem). This is due to the fact that the cluster of CPUs is only able to evaluate 10 individuals of the population in parallel, while the FPGA cluster can evaluate 58 individuals (for the ECC problem) or 116 individuals (for the other two problems) in parallel.
(2) The cost of the electricity consumed in 24 h, at the given electric rates, taking into account the power used by the processors and FPGAs, is 2.5 euros for the cluster of CPUs and only 5 euro-cents for the FPGA cluster. We have not added the cost related to the refrigeration systems to the cost of the cluster of CPUs.

5.2.2 Clusters with the same computation cost

If we assume similar computation times in both clusters (Table 6), we can use fewer nodes for the FPGA cluster, which means a lower acquisition price. Therefore, for the worst case (the XRAY problem), the price of the FPGA cluster is 3,164 euros, as against 13,200 euros for the cluster of CPUs. Also, in this case the electricity consumed in the cluster of CPUs over 24 h is still very much higher than in the FPGA cluster.

Table 5 Comparison of clusters with the same acquisition price

		XRAY	ECC	RND
FPGA	Cost of a XUP FPGA board	226 eur		
Cluster	Number of nodes	58		
	Number of HPs (NPH)	116	58	116
	Time to evaluate NPH fitness	9 s	8 s	34 s
	Cost of the cluster	13,108 eur		
	Power of de cluster	18.56 W	11.02 W	8.2 9 W
	Cost of the spent electricity for a 24 h period	0.05 eur	0.03 eur	0.02 eur
CPU	Number of nodes	10		
Cluster	Time to evaluate 10 fitness	3.11 s	39.72 s	25.61 s
	Time to evaluate NPH fitness	36.08 s	230.38 s	297.08 s
	Times FPGA cluster is faster than cluster of CPUs	4	28	9
	Cost of the cluster	13,200 eur		
	Power of de cluster	950 W		
	Times power in CPU cluster is greater than FPGA cluster	51	86	114
	Cost of the spent electricity for a 24 h period	2.56 eur		

Table 6 Comparison of clusters with the same computation time

		XRAY	ECC	RND
FPGA	Cost of a XUP FPGA board	226 eur		
Cluster	Number of nodes	14	2	7
	Number of HPs (NPH)	28	2	14
	Time to evaluate NPH fitness	9 s	8 s	34 s
	Cost of the cluster	3,164 eur	452 eur	1,582 eur
	Power of de cluster	4.48 W	0.38 W	1.0 W
	Cost of the spent electricity for a 24 h period	0.01 eur	0.001 eur	0.003 eur
CPU	Number of nodes	10		
Cluster	Time to evaluate 10 fitness	3.11 s	39.72 s	25.61 s
	Time to evaluate NPH fitness	8.71 s	7.94 s	35.85 s
	Times FPGA cluster is faster than cluster of CPUs	0.97	0.99	1.05
	Cost of the cluster	13,200 eur		
	Power of de cluster	950 W		
	Times power in CPU cluster is greater than FPGA cluster	212	2,500	949
	Cost of the spent electricity for a 24 h period	2.56 eur		

5.3 Comparing FPGA and GPU

The use of modern Graphical Processing Units (GPUs) designed for floating-point operations in many engineering problems has moved us to consider these devices to implement the same operations, in order to accelerate the same fitness functions and show the performance results against FPGAs. This way, as a first approach to the

Table 7 Comparison of FPGA and GPU

Problem	FPGA Device	FPGA times faster than GPU (CT)	FPGA times faster than GPU (TET)	FPGA times faster than CPU
XRAY	Virtex2Pro	0.79	0.95	2.28
	Spartan3-2 k	0.98	1.18	2.83
	Virtex5	7.22	8.66	20.81
ECC	Virtex2Pro	0.14	0.48	4.48
	Spartan3-2 k	0.14	0.45	4.21
	Virtex5	3.76	12.39	116.43
RND	Virtex2Pro	0.02	0.63	0.61
	Spartan3-2 k	0.02	0.47	0.46
	Virtex5	0.28	7.35	7.11

CT GPU time excluding setup time

TET GPU time including setup time

CPU Intel Core 2 DUO 6600, 1 GB RAM, 3 GHz

GPU-FPGA performance competition, we present the measures in a determined GPU comparing them to the obtained ones in the Virtex5 FPGA. Other works also tackle this competition [44].

We have used the Nvidia Quadro FX 580 [45], a GPU device with these characteristics: 512 MB memory, 32 cores, 450 MHz clock frequency. It is important to point out a power consumption of 40 watt. For programming the operations, we have considered OpenCL [46] C++ bindings.

Taking into account the kernel compilation, which is always executed at runtime, we have three values in the results: Total execution time (TET, starting at the first OpenCL instruction and stopping when the results are available), calculation time (CT, starting before executing the kernel and stopping when the results are available), and GPU setup time (ST, which can be computed by TET—CT). ST measures the time it take to setup the Open CL device and compile the kernel function (function executed by the GPU).

For the XRAY and ECC implementations, the main problem found was that ST was much bigger than CT, so we opted to put the whole evaluations loop inside the GPU. For the RND implementation only the fitness function is executed by the GPU.

Table 7 shows the comparison of FPGA and GPU results, considering the maximum number of HPs that the FPGAs are capable of hosting. In order to do a realistic comparison, the GPU time for one HP has been multiplied by the corresponding number of HPs in the FPGA. The main remark is that we obtain a good FPGA performance when the device has a large area able to fit as many HPs as be possible. On the other hand, the considered FPGAs have lower cost and power consumption than the used GPU, which is very desirable when they are applied for intensive scientific calculus in parallel computing environments.

6 Conclusions

The interest of a hardware solution based on FPGAs to solve combinatorial optimization problems lies in the possibility of accelerating the calculations by means of the parallel processors. The results obtained show that FPGAs offer better performance than CPUs for the problems considered when more than one parallel processor is used. In addition, we can say in the light of the achieved results that the FPGA performance is highly dependent on the nature of the fitness function (mainly, its ability to be parallelized in a high degree). For these reasons, the possibility of doing fitness processing by means of specifically designed reconfigurable processors in an evolutionary algorithm environment is sufficiently interesting to merit in-depth exploration of this computational alternative, which can offer better performance results, clearly surpassing those of CPUs.

Taking into account the very low power consumption of FPGA devices (less than 1 watt) in comparison to general-purpose processors (around 100 watts) and GPUs, a FPGA-based parallel computing environment emerges as a low-cost computing platform in intensive computing scenarios.

Some of the FPGAs (Virtex2Pro and Spartan3) used in this work have a lack of novelty, even though an acceptable speedup has been demonstrated. Nevertheless, modern Virtex5 device achieves the best performance rates. The results could be extrapolated to current and future technologies because the ones used in this paper are shown in a chronological manner: 2002–2003–2006/130–90–65 nm/P4.2–P4.3–Core2/Virtex2–Spartan3–Virtex5, and in this evolution the speedup stays, even improves. Also, these results show that it is possible to obtain good performances for some applications using FPGAs mounted on economical custom-designed boards.

Acknowledgments This work was partially funded by the Spanish Ministry of Science and Innovation and ERDF (the European Regional Development Fund), under the contract TIN2008-06491-C04-04 (the MSTAR project).

References

1. S. Hauck, A. DeHon (eds.), *Reconfigurable Computing, The Theory and Practice of FPGA-Based Computation* (Morgan Kaufmann, Los Altos, 2008)
2. C. Maxfield, *The Design Warrior's Guide to FPGAs: Devices, Tools and Flows* (Elsevier, Amsterdam, 2004)
3. D. Buell, T. El-Ghazawi, K. Gaj, V. Kindratenko, High-performance reconfigurable computing. Computer **40**, 23–27 (2007)
4. M. Gokhale, P. Graham, *Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays* (Springer, Berlin, 2005)
5. Z. Michalewicz, D.B. Fogel, *How to Solve It: Modern Heuristics* (Springer, Berlin, 2004)
6. C.W. Ahn, *Advances in Evolutionary Algorithms* (Springer, Berlin, 2006)
7. J. Dreo, A. Petrowski, P. Siarry, E. Taillard, *Metaheuristics for Hard Optimization* (Springer, Berlin, 2006)
8. K. Glette, J. Torresen, A Flexible On-Chip Evolution System Implemented on a Xilinx Virtex-II Pro Device, in *Evolvable Systems: From Biology to Hardware*, ed. by J. Moreno, J. Madrenas, J. Cosp (Springer, Berlin, 2005), pp. 66–75

9. R. Baraglia, et al., A parallel compact genetic algorithm for Multi-FPGA partitioning, in *9th Euromicro Workshop on Parallel and Distributed Processing* (PDP '01), pp. 113–120
10. H.E. Mostafa et al., Hardware implementation of genetic algorithm on FPGA, in *21th National Radio Science Conference* (NRSC2004) (2004), pp. C9-1-9
11. H. Emam et al., Introducing an FPGA based—genetic algorithms in the applications of blind signals separation, in *The 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications* (IWSOC'03) (2003), pp. 123–127
12. J. Wang, C.H. Lee, Complete FPGA Implemented Evolvable Image Filters, in *MICAI 2006: Advances in Artificial Intelligence*, ed. by A. Gelbukh, C.A. Reyes-Garcia (Springer, Berlin/Heidelberg, 2006), pp. 767–777
13. M.A. Vega, R. Gutiérrez, J.M. Ávila, J.M. Sánchez, and J.A. Gómez, Genetic algorithms using parallelism and FPGAs: the tsp as case study. In IEEE international conference on parallel processing, 1st workshop on parallel bioinspired algorithms (IEEE Computer Society, Oslo, Norway, 2005), pp. 573–579
14. F.C. Allaire, M. Tarbouchi, G. Labonté, G. Fusina, FPGA implementation of genetic algorithm for UAV real-time path planning. J. Intell. Robot. Syst **54**, 495–510 (2009)
15. John R. Koza, Stephen L. Bade, Martin A. Keane, Jeffrey L. Hutchings, Rapidly reconfigurable field-programmable gate arrays for accelerating fitness evaluation in genetic programming, *Genetic Programming Conference* (1997), pp. 121–131
16. Q. Qiu, D. Burns, P. Mukre, Q. Wu, Hardware acceleration of multi-deme genetic algorithm for the application of DNA codeword, in *9th annual conference on Genetic and evolutionary computation* (2007), pp. 1349–1356
17. K.D. Underwood, K.S. Hemmert, Closing the gap: CPU and FPGA Trends in sustainable floating-point BLAS performance, in *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines* (FCCM'04) (2004), IEEE Computer Society
18. B. Sukhwani, M. Chiu, M.A. Khan, and M.C. Herbordt, Effective floating point application on FPGAs: examples from molecular modeling, in *High Performance Embedded Computing* (HPEC 2009) (2009), Lexington, MA, USA
19. C. Pedraza, J. Castillo, J. Martínez, P. Huerta, J. Bosque, and J. Cano (2010, March). Genetic algorithm for Boolean minimization in an FPGA cluster. J. Supercomput. [Online]. Available: http://www.springerlink.com/content/654l37k832678t60
20. F. Belleti et al., Ianus: An Adaptive FPGA Computer. Comput Sci. Eng. **8**, 41–49 (2006)
21. C. Patterson, S. Ellingson, B. Martin, K. Deshpande, J. Simonetti, M. Kavic, and S. Cutchin, *Searching for Transient Pulses with the ETA Radio Telescope, ACM Transactions on Reconfigurable Technology and Systems1* (2009), pp. 1–20
22. M. Yoshimi, Y. Nishikawa, M. Miki, T. Hiroyasu, H. Amano, and O. Mencer, A performance evaluation of CUBE: one-dimensional 512 FPGA cluster, in *Reconfigurable Computing: Architectures, Tools and Applications* (Ed: Springer, 2009), pp. 372–381
23. V. Sriram, M. Leeser (eds.), *FPGA Supercomputing Platforms, Architectures, and Techniques for Accelerating Computationally Complex Algorithms* (Hindawi, Cairo, 2009)
24. R. Hamming, Error detecting and error correcting codes. Bell Syst. Tech. J **29**, 147–160 (1950)
25. K. Dontas, K.D. Jong, Discovery of maximal distance codes using genetic algorithms, in *2nd International IEEE Conference on Tools for Artificial Intelligence* (IEEE Computer Society, 1990), pp. 905–811
26. A.E. Gamal, L. Hemachandra, I. Shperling, V. Wei, Using simulated annealing to design good codes. IEEE Trans. Inf. Theory **33**, 116–123 (1987)
27. E. Alba, J.F. Chicano, *Solving the error correcting code problem with parallel hybrid heuristics* (ACM Symposium on Applied Computing, ACM, Nicosia, Cyprus, 2004), pp. 985–989
28. P. Calegari, F. Guidec, P. Kuonen, D. Kobler, Parallel island-based genetic algorithm for radio network design. J. Parallel Distrib. Comput. **47**, 86–90 (1997)
29. P. Calegari, F. Guidec, P. Kuonen, Combinatorial optimization algorithms for radio network planning. J. Theor. Comput. Sci. **263**, 235–265 (2001)
30. E. Alba, *Evolutionary Algorithms for Optimal Placement of Antennae in Radio Network Design* (IEEE NIDISC, IEEE, Santa Fe, New Mexico, USA, 2004), pp. 168-174
31. S. Khuri, T. Chiu, *Heuristic algorithms for the terminal assignment problem* (ACM Symposium on Applied Computing, 1997), pp. 245–251
32. P. Thompson, D.E. Cox, J.B. Hastings, Rietveld refinement of Debye-Scherrer synchrotron X-ray data from Al2O3. J. Appl. Cryst **20**, 79–83 (1987)

33. T.H.D. Keijser, E.J. Mittemeijer, H.C.F. Rozendaal, The determination of crystallite-size and lattice-strain parameters in conjunction with the profile-refinement method for the determination of crystal structures. J. Appl. Cryst. **16**, 309–316 (1983)
34. S. Enzo, G. Fagherazzi, A. Benedetti, S. Polizzi, A profile-fitting procedure for analysis of broadened X-ray diffraction peaks. I. Methodology. J. Appl. Cryst **21**, 536–542 (1988)
35. F. Sánchez-Bajo, F.L. Cumbrera, The use of the Pseudo-Voigt function in the variance method of X-ray line-broadening analysis. J. Appl. Cryst. **30**, 427–430 (1997)
36. J. Gomez-Pulido, *Hardware Designs for Fitness Performance* (2010) Available: http://arco.unex.es/fitness_performance
37. K. Ramamritham, K. Arya, System software for embedded applications, in *17th International Conference on VLSI Design* (2004), pp. 22–24
38. V. Kindratenko, D. Pointer, D. Raila, C. Steffen, Comparing CPU and FPGA Application performance, Tech. Report, 2006
39. D.B. Thomas, L. Howes, W. Luk, A Comparison of CPUs, GPUs, FPGAs and Massively parallel processor arrays for random number generation, in *ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (ACM New York, NY, USA, Monterey, California, USA, 2009), pp. 63–72
40. S. Asano, T. Maruyama, Y. Yamaguchi, Performance comparison of FPGA, GPU and CPU in image processing, in *IEEE 19th International Conference on Field Programmable Logic and Applications* (Prague, Czech Republic, 2009), pp. 126–131
41. D.A. Patterson, J.L. Hennessy, *Computer Organization and Design—The Hardware/Software Interface* (Morgan Kaufmann, Los Altos, 2009)
42. Intel Processors: http://www.intel.com/products/processor_number, 2009
43. Iberdrola Electric Rates: http://www.iberdrola.es, 2009
44. S. Chey, J. Liz, J.W. Sheaffery, K. Skadrony, J. Lach, Accelerating compute-intensive applications with GPUs and FPGAs, in: *IEEE Symposium on Application Specific Processors*, 2008, pp. 101–107
45. NVIDIA Quadro FX 580 Datasheet, NVIDIA Corporation, 2009
46. D. Kirk, W.-M. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach* (Morgan Kaufmann, Los Altos, 2010)