


RESEARCH ARTICLE

Open Access



# Fine-grained parallelization of fitness functions in bioinformatics optimization problems: gene selection for cancer classification and biclustering of gene expression data

Juan A. Gomez-Pulido<sup>1\*</sup> , Jose L. Cerrada-Barrios<sup>1</sup>, Sebastian Trinidad-Amado<sup>1</sup>, Jose M. Lanza-Gutierrez<sup>1</sup>, Ramon A. Fernandez-Diaz<sup>2</sup>, Broderick Crawford<sup>3,4</sup> and Ricardo Soto<sup>3,5,6</sup>

## Abstract

**Background:** Metaheuristics are widely used to solve large combinatorial optimization problems in bioinformatics because of the huge set of possible solutions. Two representative problems are gene selection for cancer classification and biclustering of gene expression data. In most cases, these metaheuristics, as well as other non-linear techniques, apply a fitness function to each possible solution with a size-limited population, and that step involves higher latencies than other parts of the algorithms, which is the reason why the execution time of the applications will mainly depend on the execution time of the fitness function. In addition, it is usual to find floating-point arithmetic formulations for the fitness functions. This way, a careful parallelization of these functions using the reconfigurable hardware technology will accelerate the computation, specially if they are applied in parallel to several solutions of the population.

**Results:** A fine-grained parallelization of two floating-point fitness functions of different complexities and features involved in biclustering of gene expression data and gene selection for cancer classification allowed for obtaining higher speedups and power-reduced computation with regard to usual microprocessors.

**Conclusions:** The results show better performances using reconfigurable hardware technology instead of usual microprocessors, in computing time and power consumption terms, not only because of the parallelization of the arithmetic operations, but also thanks to the concurrent fitness evaluation for several individuals of the population in the metaheuristic. This is a good basis for building accelerated and low-energy solutions for intensive computing scenarios.

**Keywords:** Biclustering, Cancer classification, FPGA, Parallelism, Floating-point arithmetic, Metaheuristics, Fitness function

**Abbreviations:** ACO, Ant colony optimization; ACPI, Advanced configuration and power interface; ANN, Artificial neural networks; CMOS, Complementary metal-oxide-semiconductor; CPU, Central processing unit; DE, Differential evolution; DSP, Digital signal processor; EA, Evolutionary algorithm; FPGA, Field programmable gate array; GA, Genetic algorithm; GBC, Geometric biclustering; GPU, Graphical processing unit; HDL, Hardware description language; HPRC, High-performance reconfigurable computing; MSR, Mean squared residue; PSO, Particle swarm optimization; RC, Reconfigurable computing; SVM, Support vector machine

\*Correspondence: jangomez@unex.es

<sup>1</sup>Department of Technologies of Computers and Communications, University of Extremadura, Polytechnic School, Campus Universitario s/n, 10003 Caceres, Spain

Full list of author information is available at the end of the article

## Background

Bioinformatics is an area where we can find many large combinatorial optimization problems [1]. The high size of the space of solutions causes these problems can not be tackled by means of exact searching techniques, which require an excessive computational effort. In these cases, the usual way of obtaining optimal solutions is to consider metaheuristics [2] and particularly Evolutionary Algorithms (EAs) [3]. Nevertheless, even these algorithms can be slow for complex problems, demanding more hardware resources based on current general-purpose processors or Central Processing Units (CPUs). If we identify what part of the algorithm takes more time to be computed, a hardware coprocessor specifically designed to accelerate this function is a direct solution to further speed up the performance. In this sense, the fitness function is a simple but critical operation involved in the metaheuristics. Most of the computing time of the algorithm that solves the optimization problem may be spent running the fitness function, although it could mean a small part of the code.

The core of this work deals with the hardware-level parallelization of the fitness functions used in two bioinformatics problems: gene selection for cancer classification and biclustering of gene expression data. The reason for designing fitness hardware accelerators is twofold. On the one hand, every fitness function is applied to each individual of a population in many bio-inspired metaheuristics; this fact allows us to parallelize the computation of the fitness evaluation phase if we place several copies of the same fitness hardware implementation. On the other hand, fitness functions are usually formulated by means of floating-point arithmetic equations that can involve many operation steps; this way, parallelization of some of these steps using repeated units of the same floating-point operator increases the performance of the design.

Both reasons represent two levels of parallelism: in the bottom, a fine-grained parallelization of the fitness equation; in the top, a fast computation of the fitness evaluation phase applying replicated fitness units in parallel to several individuals of the population. We focused our research mainly on the fine-grained parallelization of the fitness formulation, although on-chip concurrent fitness evaluation has been explored as well. Figure 1 illustrates these considerations, comparing usual CPU sequential programming to custom on-chip parallel systems. We can accelerate the computation of the fitness phase making good use of parallelism: replicated fitness functions working in parallel at the top-level, and parallel computation of the fitness equation at the bottom-level. We can observe that CPU requires sequential steps not only for the evaluation of the fitness of each individual, but for the calculation of the fitness equation.

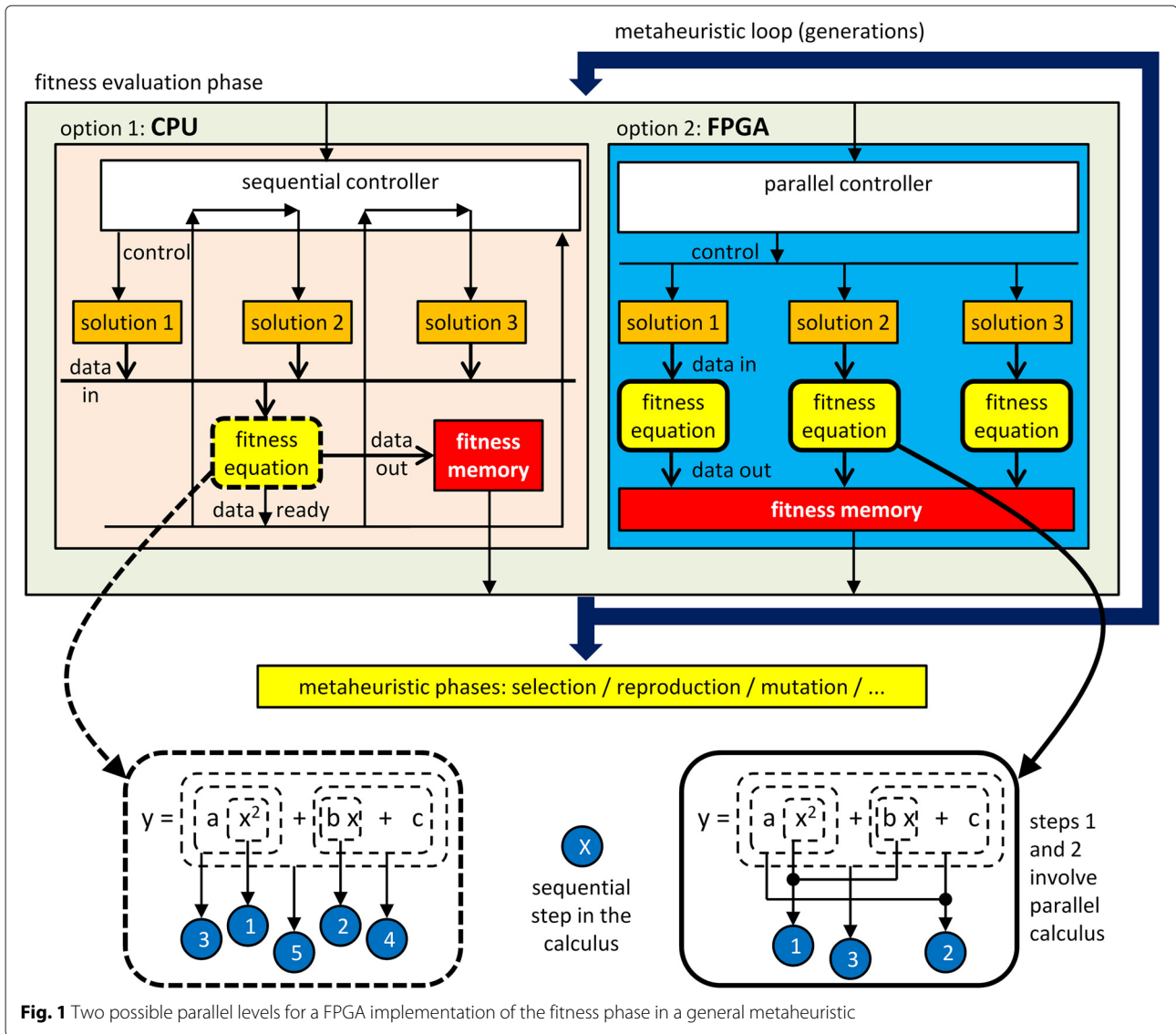
The hardware implementation of fitness equations is made easier thanks to Hardware Description Languages

(HDLs) and Field Programmable Gate Array (FPGA) devices [4]. The FPGA technology favoured the rise of a computing domain that combines software flexibility with hardware performance exploiting the parallel paradigm: Reconfigurable Computing (RC) [5]. This way, a fitness function carefully designed can surpass the CPU performance in similar experimental conditions, as RC has demonstrated in many applications [6]. In addition, we decide on FPGAs instead of other competitive technologies as Graphical Processing Units (GPUs) since FPGAs usually provide better performance and lower power consumption than GPUs [7].

Reconfigurable computing has been successfully applied to many bioinformatics problems, because they have a high parallelism degree. Knowing how to make the most of this parallelism, we can obtain speedups and energy savings needed for intensive computing or real-time applications. In this area, we can find FPGA implementations for DNA matching based on the BLAST algorithm [8], Bowtie short-read mapping [9], epistasis detection [10], molecular modeling [11], and many other algorithms involved in sequence comparison, multiple sequence alignment, RNA and protein secondary structure prediction, gene prediction and phylogenetic tree computation [12], among many others. Nevertheless, these works are usually focused on solving specific problems, dealing with their special characteristics and constraints. Contrary to these approaches, our work tries to get a wide insight into important aspects to take into account when designing accelerators.

This way, our main contribution in this paper is to demonstrate that the fine-grained parallelization of fitness functions based on floating-point arithmetic can surpass the performance given by CPUs, in time and power terms, when they are massively used by metaheuristics for solving large combinatorial optimization problems in bioinformatics. The conclusions of our work can be applied in general to similar cases, because of the representativeness of the fitness functions we have chosen. For this purpose, we have selected two specific fitness functions used in the above mentioned optimization problems by two reasons: on the one hand, there is not enough information about their implementation in FPGAs in the existing literature; on the other hand, they provide different computational workloads and parallelization levels because of their floating-point arithmetic formulations, being representative formulations of other similar functions widely used in bioinformatics.

The hardware implementation of the fitness function can be used as a coprocessor of an embedded CPU running the metaheuristic in the same FPGA. Nevertheless, the need for scalability that large and real-world applications require, and the metaheuristic request of handling many individuals of the population



**Fig. 1** Two possible parallel levels for a FPGA implementation of the fitness phase in a general metaheuristic

in parallel, make it necessary to consider computing systems consisting of several FPGAs in multicore architectures. This coarse-grained parallelization belongs to the High-Performance Reconfigurable Computing (HPRC), a promising paradigm that exploits the possibilities of FPGAs [13, 14], although it requires to design according to computing models based on specific communication and data-handling techniques [15]. Nevertheless, if we want to develop a computing system based on such large FPGA platforms, the first and mandatory step is to know if the unit to be massively replicated (in our case, the fitness function) is able to give enough speedup with regard to usual CPUs. This is the reason why our research is focused on a worthwhile fine-grained parallelization of the fitness function, since it is the basis for a success scalability that is left as future development.

Summarizing, our proposal presents the performance from a computational perspective. Other performance features closer to the specific bioinformatics problems only can be tackled by the corresponding algorithmic methods and software packages, which are out of the scope of this work.

**Related work**

As we pointed out in the previous section, bio-inspired and evolutionary optimization algorithms are very appropriate to be parallelized, not only by applying repeated fitness hardware units in parallel on several individuals of the population, but parallelizing other important parts. For example, the intrinsic parallelism in popular Genetic Algorithms (GAs) [16] allows better speedups. In this line, FPGAs have been successfully applied to parallelize many metaheuristics and optimization algorithms, like

Differential Evolution (DE) [17], Particle Swarm Optimization (PSO) [18], Artificial Neural Networks (ANN) [19], and Ant Colony Optimization (ACO) [20], among many others.

The high performance cost of the fitness evaluation phase in relation to the overall computing time of the metaheuristic is a well-studied fact in the literature. Fitness evaluation can take up to 95 % of the total execution time in genetic programming [21] or 64 % in GAs with evolutionary mapping [22]. In general, many works have demonstrated that the execution time of the applications will mainly depend on the execution time of the fitness function [23, 24].

The above considerations move us to implement the fitness functions in hardware to enhance the system performance. These functions have been accelerated by means of FPGA devices in genetic programs for financial markets [25], spatial image filters [26], filtered image signals [27], test cases [28], and many other engineering applications.

The first bioinformatics problem in our study is gene selection for classification of high dimensional Microarray data in cancer disease. This optimization problem has been studied using mainly GAs and Support Vector Machines (SVMs), where the GA is used to evolve gene subsets whose fitness is evaluated by a SVM classifier. In this line, there are approaches based on single objective [29] and multi-objective [30] points of view. Nevertheless, we have not found any FPGA implementation of fitness functions associated to this problem. Therefore, we offer novel insight into its hardware parallelization.

The second optimization problem considered in our research deals with biclustering of gene expression data, which has been tackled by means of custom evolutionary algorithms [31, 32]. We have not found any FPGA implementation of the fitness function as it is formulated in these works. Nevertheless, FPGAs have been applied in a related work, in order to accelerate the Geometric Biclustering (GBC) algorithm [33]; in this work, we compared the FPGA implementation with multi-core CPU and GPU architectures, and found out that FPGA achieved higher speedup for large microarrays, as well as lower power consumption.

### Two case studies in bioinformatics

We have tackled the implementation of the two above mentioned bioinformatics problems following the same strategy: first, we design a fine-grain parallel circuit that implements the fitness function; then, we measure the speed-up with regard to current general-purpose processors for just one fitness evaluation; finally, we estimate the performance when several fitness circuits evaluate individuals in parallel, taking into account the area constraints for a single FPGA. This approach allows us to apply the fitness circuits as coprocessors of an embedded

processor that drives different optimization algorithms. This methodology is similar to other studies, as [34, 35], where a single FPGA contains multiple instances of fitness circuits to evaluate possible solutions in parallel, together with the optimization algorithm driven by an embedded processor.

There are many other bioinformatics problems involving metaheuristics with fitness functions similar to these two cases, specifically with regard to the floating-point arithmetic [36, 37]. This way, analyzing the FPGA implementation of the two case studies can contribute to expect good computing speedups in other works.

### Gene selection for cancer classification

The analysis of microarray-based gene expression allows us to compare between the gene expression levels of cancerous and normal cells, in order to select the genes under suspicion [38]. These genes are useful for cancer classification, but hard to be selected when the number of genes ( $M$ ) and samples ( $N$ ) are very high, shaping a combinatorial optimization problem.

A common approach to face this challenge consists in selecting a subset of suspicious genes for cancer classification. This is the basis of many metaheuristics where the individuals of the population are gene subsets. We have considered a fitness function given by (1), where  $x$  is the subset,  $A(x)$  is the leave-one-out-cross-validation accuracy provided by a classifier,  $R(x)$  is the number of selected genes in the subset, and  $w_1$  and  $w_2$  are weights for the accuracy level and the number of selected genes, respectively [30]. This fitness function must be maximised by the metaheuristics in order to find an optimal gene subset.

$$F(x) = w_1 A(x) + w_2 \frac{M - R(x)}{M} \quad (1)$$

The top-level circuit to test the fitness function (Additional file 1: Figure S1) is composed of  $NF$  instances of the fitness circuit,  $NC$  instances of a floating-point comparator, and a controller that drives and parallelizes the operations involved in  $F$ . The value of  $NF$  depends on the FPGA area.

The mission of the controller is to handle the different steps of the test process, which follows this scheme:

1. The controller simultaneously sends different subsets to the fitness units, together with a start instruction and other values involved in the fitness calculation.
2. The fitness units start to compute  $F$  in parallel for each subset. The calculation in each unit is parallel too.
3. Once all the  $NF$  units have calculated the fitness values, they are sent in parallel by pairs to  $NC = NF/2$  floating-point comparators.

4. The comparators determine the highest values of the fitness pairs. Once all the comparisons have finished, the best values are compared again by pairs, this time by means of  $NC/2$  comparators.
5. The comparison process continues up to reach the last pair of higher values, where the highest one is given back to the controller.

The fitness circuit implements the arithmetic operations involved in  $F$ , some of them in parallel. The architecture of the fitness unit (Additional file 1: Figure S2) is composed of several arithmetic modules and a fitness controller. The fitness controller drives the arithmetic operations according to (1), where three operations are performed in parallel:  $w_1A(x)$ ,  $w_2/M$  and  $M - R(x)$ . This architecture needs three floating-point arithmetic operators (adder, multiplier and divider) and an integer to float converter. The fitness controller supplies the operands to the arithmetic modules and receives the results. Once the calculation of  $F$  has been completed, the fitness controller gives it back to the controller.

#### Biclustering of gene expression data

This problem deals with numerical matrices that represent information extracted from microarray data. These matrices can be built using clustering or biclustering methods [39]. Clustering methods gather together genes with a similar behaviour under all the experimental conditions, using algorithms based on genes similarity, whereas biclustering methods find subsets of genes with the same behaviour under a subset of experimental conditions.

A general bicluster is represented by a matrix  $B$  of  $I$  rows (number of experimental conditions) and  $J$  columns (number of genes), where the element  $b_{ij}$  is the expression level of the gen  $j$  under the experimental condition  $i$ .

Since biclustering is more complex than clustering, several evolutionary algorithms have been applied in order to find biclusters. These algorithms consider as fitness function a measure for assessing the quality of biclusters. One usual measure is the Mean Squared Residue (MSR), that provides lower values for better biclusters. The MSR value is calculated following these steps:

1. Calculation of the means  $biJ$  of each row  $i$  (2).
2. Calculation of the means  $blj$  of each column  $j$  (3).
3. Calculation of the mean  $bIJ$  of the entire matrix (4).
4. Calculation of the residue  $r_{ij}$  of each matrix element (5).
5. Calculation of the MSR (6).

$$b_{ij}[i] = \frac{\sum_{j=0}^{J-1} b_{ij}}{J} = \frac{sum\_biJ_i}{J} \quad (2)$$

$$b_{ij}[j] = \frac{\sum_{i=0}^{I-1} b_{ij}}{I} = \frac{sum\_blj_j}{I} \quad (3)$$

$$bIJ = \frac{\sum_{i=0}^{I-1} \sum_{j=0}^{J-1} b_{ij}}{J} = \frac{sum\_bIJ}{I \cdot J} \quad (4)$$

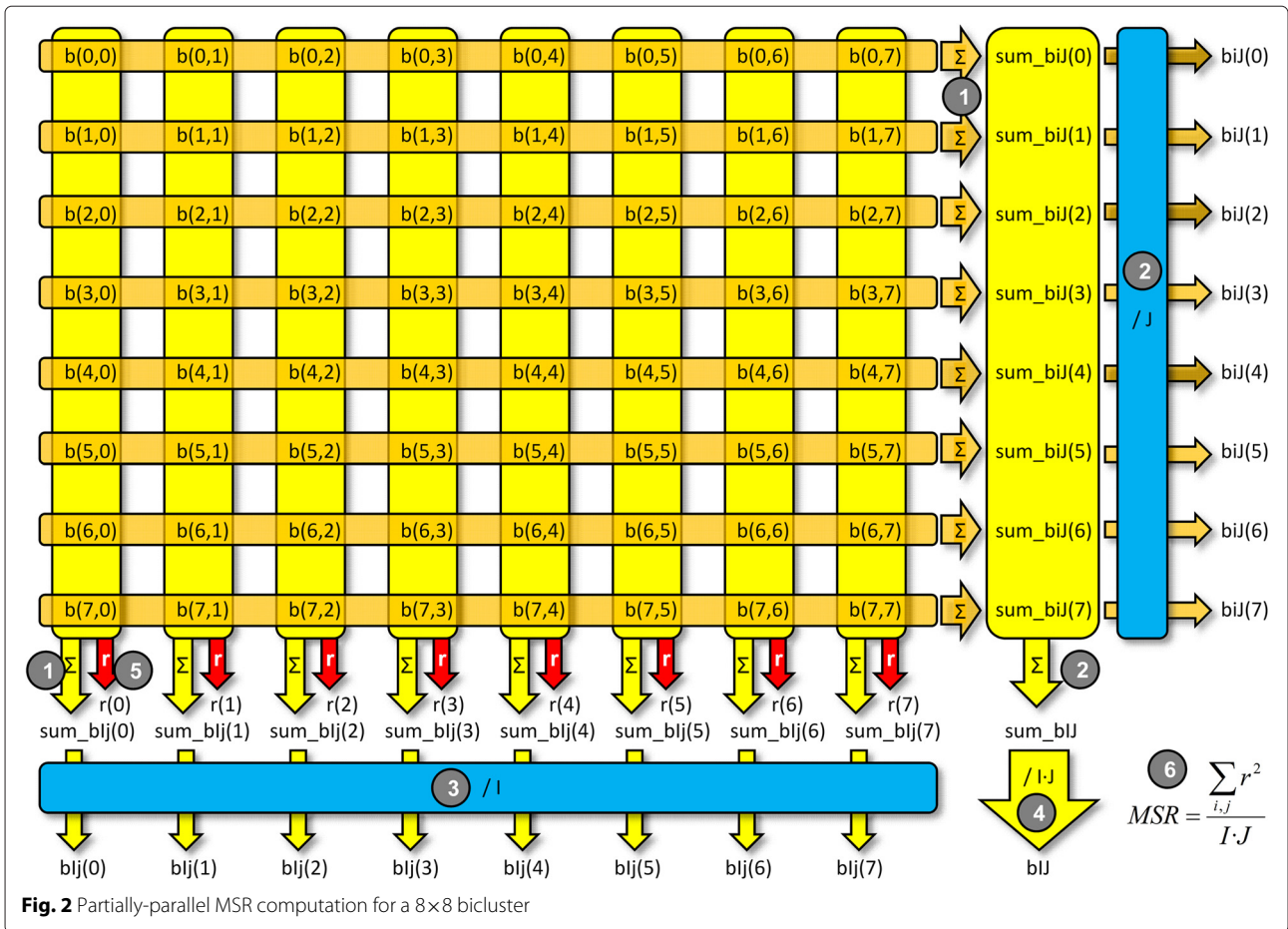
$$r_{ij} = b_{ij} - biJ_i - blj_j + bIJ \quad (5)$$

$$MSR = \frac{\sum_{i=0}^{I-1} \sum_{j=0}^{J-1} (r_{ij})^2}{I \cdot J} \quad (6)$$

This procedure is highly parallelizable. There are different ways to parallelize the calculation of MSR, according to the experimental constraints: the more resources we have, the more parallelization we can achieve. Since the parallelism comes basically from the use of replicated circuits of the floating-point arithmetic operators, the FPGA device can host different number of these units depending on two factors: the specific FPGA device (family and model) and the size of the bicluster. Due to this reason, we have considered two different parallelization models to compute MSR.

We name the first model as *MSR partially parallelized*. This is a procedure useful for bigger matrices or FPGA devices with lower area, where we can only use a limited number of repeated circuits for the arithmetic operators. This procedure involves more sequential steps than in the case where we have as many multipliers as elements  $b_{ij}$  in the matrix. This way, the computation of MSR follows six sequential steps, each of them composed of parallel tasks, as Fig. 2 shows an example of a  $8 \times 8$  bicluster:

1. The elements  $b_{ij}$  of each row and column are added in parallel, obtaining at the same time the values of  $sum\_biJ_i$  and  $sum\_blj_j$ .
2. The sum  $sum\_blj$  of all the elements  $b_{ij}$  of the matrix (adding the values of  $sum\_biJ$  for all the rows) is obtained in parallel together with the values of  $biJ_i$  (obtained dividing  $sum\_biJ_i$  by  $J$ ) according to (2).
3. The values of  $blj_j$  are obtained in parallel dividing the corresponding  $sum\_blj_j$  by  $I$ , according to (3).
4. The value of  $bIJ$ , according to (4), is calculated dividing  $sum\_bIJ$  by  $I \cdot J$ .
5. The values of  $r_{ij}$ , according to (5), are calculated in parallel by rows, but sequentially by columns, taking into account that the number of parallel floating-point multipliers is limited.
6. Finally, the value of MSR, according to (6), is obtained parallelizing the calculation of  $r^2$ .



The MSR fully parallelized model parallelizes the MSR computation in a higher grade. This procedure can be applied to large FPGA devices or smaller matrices. In this case, the MSR calculation follows five sequential steps, each of them also composed of parallel tasks, as Fig. 3 shows for an example of a 4×4 bicluster:

1. The first step is the same as in the MSR partially parallelized model: calculation of  $sum\_bij_i$  and  $sum\_blj_j$ .
2. Now we increase the parallelism with regard to the first model, calculating in parallel  $sum\_bIJ$ ,  $bij_i$  and  $blj_j$ .
3. This step corresponds with the fourth step in the first model: calculation of  $bIJ$ .
4. Now we can calculate  $r_{ij}$  in a fully parallel way, because we have more parallel floating-point multipliers.
5. The last step calculates MSR as the previous model does.

The top-level circuit that measures the MSR performance (Additional file 1: Figure S3), just like the fitness

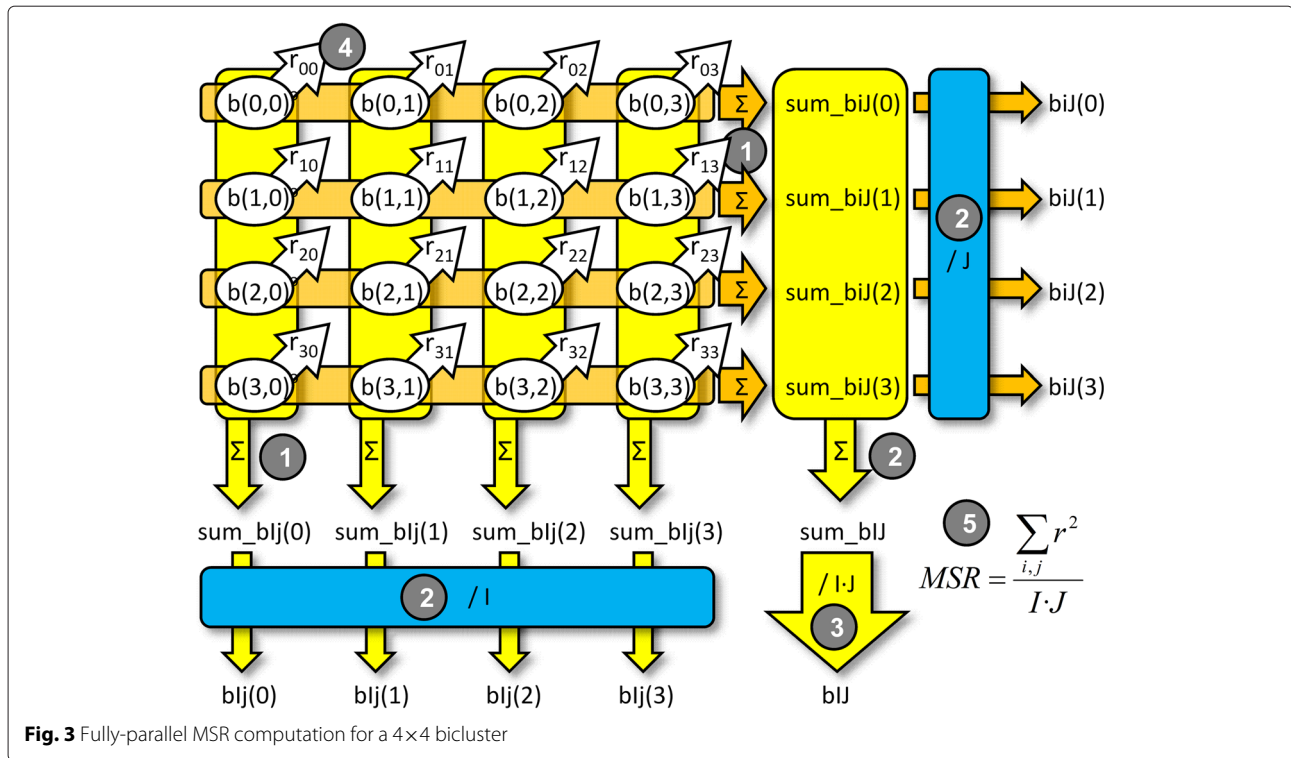
function for the first bioinformatics problem, is composed of  $NF$  instances of the fitness circuit,  $NC$  instances of a floating-point comparator, and a controller. The value of  $NF$  and the corresponding  $NC = NF/2$  also depend on the FPGA area.

The controller and the fitness circuits have different implementations according to the parallelization model and the bicluster size. The implementation version is identified by one letter ( $f$  for the partially parallelized model, and  $a$  for the fully parallelized one) followed by the matrix size. In addition, the number of fitness and comparator units is specified for the controller. For example, *controller-f16x8-NF6-NC3* denotes the circuit implementation for a bicluster of 16 experimental conditions and 8 genes driven by the partially parallelized model using 6 parallel fitness units; in this case, the fitness circuit associated with this controller is identified as *msr-f16x8*.

The architecture of the fitness circuit (Additional file 1: Figure S4) may contain different number of adders, multipliers, dividers and integer-to-floating point converters, according to the implementation version. Each implementation version takes into account specific parallelization and resource use. For example, the design

$$MSR = \frac{\sum_{i,j} r^2}{I \cdot J}$$





*controller-f8x8-NF4-NC2* hosts 2 comparators and 4 fitness units of type *msr-f8x8*, each of them containing 8 adders, 8 multipliers, 8 dividers and 8 converters, whereas a *controller-a4x4-NF4-NC2* circuit counts the same number of comparators and fitness units of type *msr-a4x4*, each of them containing 16 adders, 16 multipliers, 16 dividers and 16 converters. Obviously, the designs following the fully-parallelized model need much more hardware resources than the partially-parallelized model, even for smaller biclusters.

**Results**

This section summarizes the tools, hardware resources, and implementation keys from which the results were obtained.

**Design tools**

We have designed the fitness units, controllers and assistant circuits using programming languages and tools specifically used for designing with reconfigurable hardware. The main cores were programmed using VHDL hardware description language [40]. This is a very efficient and known language, specially when we are programming at the register-transfer level, allowing to program algorithms abstracting away the hardware as far as it is possible.

On the other hand, we have used Xilinx ISE 14 software suite [41] for the simulation, synthesis and

implementation of the top-level circuits. This suite contains two important tools: on the one hand, CORE Generator System tool was used for generating the circuits for the floating-point arithmetic operators; on the other hand, ISim simulator was used for testing the top level circuit and measuring the time responses, very useful to calculate the speedups of the FPGAs with regard to CPUs.

The design methodology follows some steps, starting from the programming of the circuits using VHDL and CORE Generator tool. In this step is mandatory to do the maximum parallelization effort in order to design an efficient architecture. Once built the codes, the synthesis and implementation step allows obtaining the minimum clock frequency for a determined FPGA device. Using this information, a VHDL testbench customized with the corresponding clock period can simulate the top level design using ISim, obtaining the time response of the circuit, which will be used to calculate the FPGA speedup.

**Hardware resources**

Table 1 shows the hardware used for the experiments: FPGA devices for implementing the fitness circuits and general-purpose CPUs for comparing the performance results.

The selected Xilinx FPGA devices offer a representative range of features, including the low-cost Spartan6 (xc6slx150), the high-performance Virtex6 (xc6vlx550t)

**Table 1** Hardware resources

Devices	Features			
FPGAs:	Technology	Logic cells	DSP slices	RAM blocks
xc5vlx330-1ff1760	65nm	331,776	192	10,368 kB
xc6vlx550t-2ff1759	40nm	549,888	864	22,752 kB
xc6slx150-3fgg676	45nm	147,443	180	4,824 kB
CPU:	Technology	GHz		
Core2-E6750	65nm	2.6		
i7-950	45nm	3.07		
i5-2430	32nm	2.4		
i7-2600	32nm	3.4		

and the balanced Virtex5 (xc5vlx330). These devices may be characterized by four important features that describe the process technology (Complementary Metal-Oxide-Semiconductor -CMOS- depth in nanometers), the number of logic cells (as indicator of the area available to host the circuits), the number of internal Digital Signal Processor (DSP) slices (related to the speed of the floating-point arithmetic operators) and the number of memory blocks (useful to handle the circuit data).

We measured the performance of our three FPGA devices with the post-placement and routing simulation tool provided by the implementation environment. The validation of the results consisted in comparing the simulation times of the Virtex5 device with those measured with custom circuits on a prototyping board that hosted the xc5vlx330 device: Xilinx University Program Virtex5 Development Kit. Since both times were almost equal, we can approve the simulation results corresponding to the other FPGA devices.

In order to establish a valid FPGA vs CPU comparison, and properly analyze the performances, we should consider the use of contemporary devices with similar technologies. This reason led us to use several processors of different CMOS technologies and clock frequencies, as we can see in Table 1.

### Implementation keys

We designed custom circuits to test the performance of the fitness function instead of using embedded processors because these ones take up an area that, otherwise, would be useful for hosting more parallel fitness circuits.

Each synthesis was repeated several times following different strategies in order to obtain the highest clock frequency. On the one hand, we considered three optimization synthesis profiles: default, timing performance with physical synthesis, and timing performance without input/output blocks packing; other synthesis profiles were discarded because of their worse results. On the other hand, we have tested two possibilities when it comes to

synthesizing the floating-point arithmetic operators by CORE Generator: using internal DSPs or logic blocks in the architecture optimization. If we consider DSPs, the performance can be better, but the limited number of DSPs forces us to consider digital logic if we want to have more parallel units, involving more area consumption; this tradeoff between number and performance of parallel operators must be evaluated in each case.

This way, each design was synthesized up to 6 times (according to the 3 synthesis profiles and the 2 possibilities of using DSPs in the operator circuits), recording the best result among the obtained ones. For example, for the fitness function in the gene selection for cancer classification problem, we tested 6 cases (8, 16, 32, 64, 128 and 256 parallel fitness units); therefore, 6 cases x 3 synthesis profiles x 2 operator optimizations = 36 synthesis experiments were performed. Depending on the *NF* value considered, the synthesis took from 1 hour to 2 days, also according to the processor used among those listed in Table 1. This means many days running synthesis processes. For the fitness function in the biclustering of gene expression data problem, 8 cases corresponding to different matrix sizes and parallelizing strategies were tested (f4x4, f8x8, f16x8, f16x16, f30x50, f32x64, a4x4, a5x5), totalizing 48 syntheses.

Each synthesis reports interesting data with regard to the scalability and performance of the fitness circuits:

1. Area occupation. Several indicators (slice registers, slice Look-Up-Tables and occupied slices) allow us to calculate the number of circuits that we can replicate in the same FPGA device in order to work in parallel. Depending on the values returned by these indicators and the FPGA family and model, a different number of such circuits can be considered.
2. Timing performance. The value of the maximum frequency (MHz) (that corresponds to the minimum clock period in nanoseconds) allows us to determine the time to process the fitness function; if we consider *NF* parallel units of the fitness circuit, the time to process the different solutions is equal to that time.
3. Power consumption. Nowadays, it is very important to design energy-aware circuits in order to minimize operation costs when solving problems that involve massive computations along the time. The synthesis process tells us the power (watts) consumed by the fitness circuits.

### Discussion

Among the many data returned by the synthesis processes, we analyze mainly the timing reports, since they provide the speedup of FPGA versus CPU (of course, we have checked the numerical results are the same in both FPGA and CPU implementations). We understand



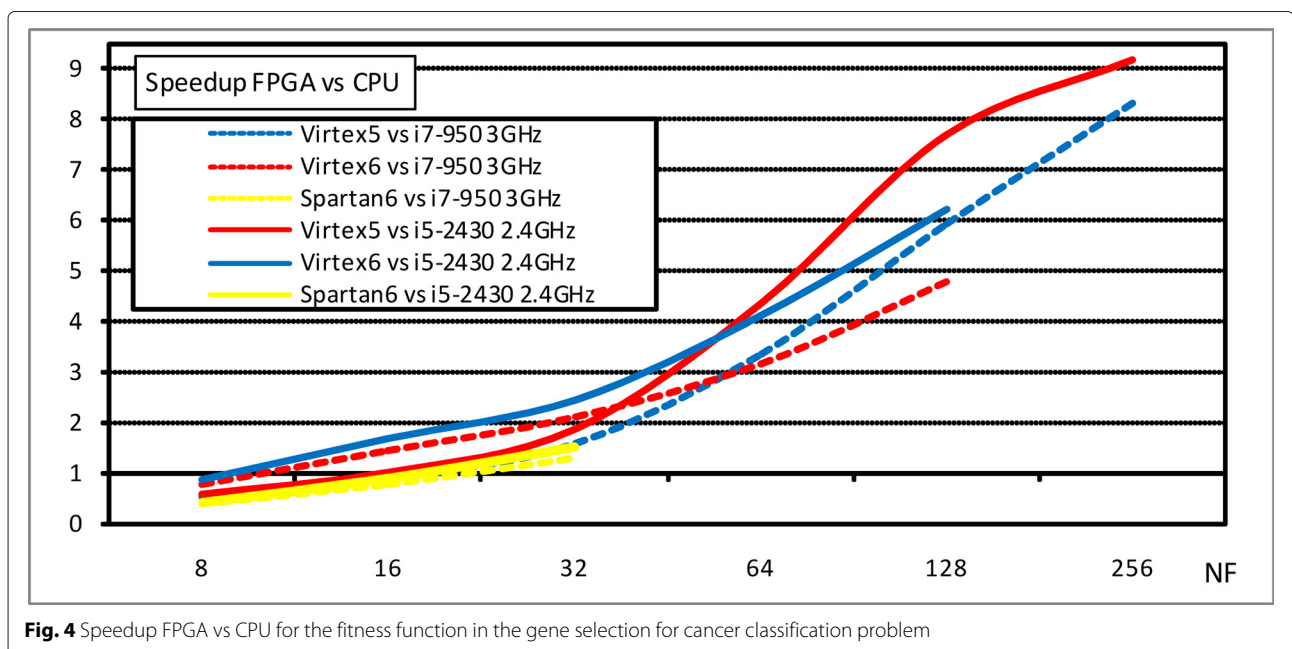
by timing performance the reciprocal of the computing time  $T$  [42]. To compare the performance of FPGAs and processors, we say that the speedup of FPGA versus CPU is  $T_{CPU}/T_{FPGA}$ . Hence, a speedup greater than one means that FPGA is faster than CPU; otherwise the processor wins. It is important to realize that both values,  $T_{CPU}$  and  $T_{FPGA}$ , measure the same number of fitness evaluations; in the first case, using a loop of sequential computations, whereas the second case considers a parallel computation of  $NF$  fitness circuits.

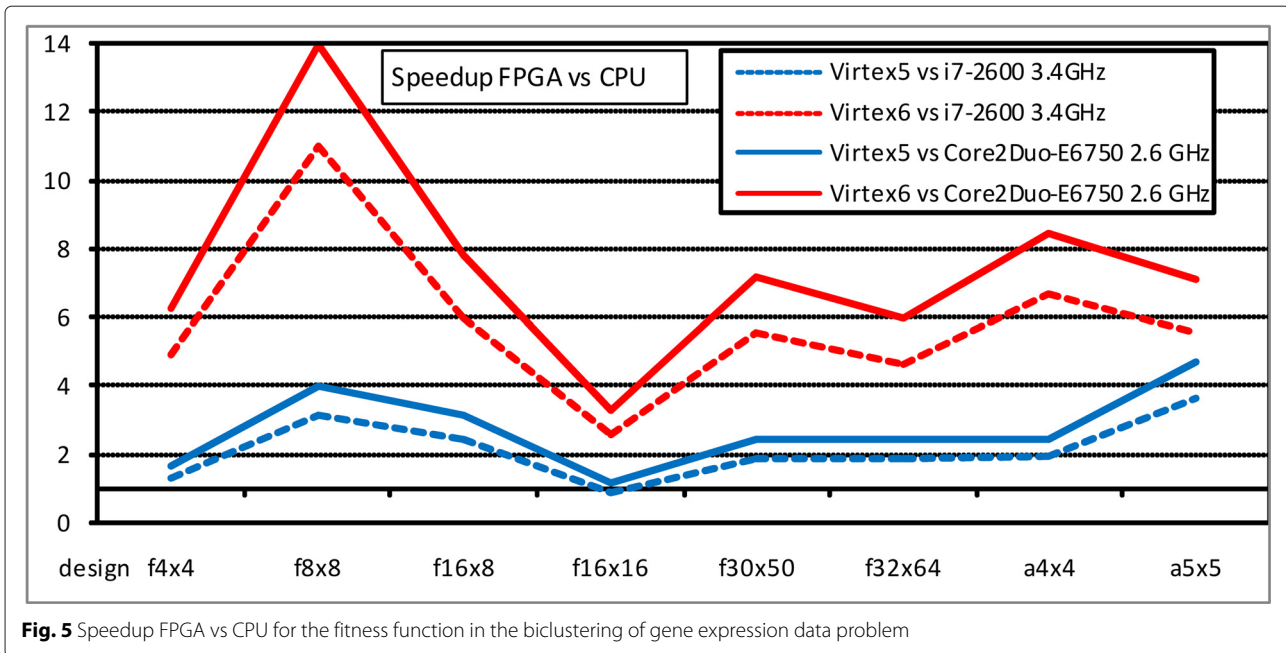
According to this speedup definition, and taking into account the maximum number of parallel fitness circuits that can operate in parallel in the same FPGA, Fig. 4 shows that FPGAs are much faster than CPUs computing the fitness phase in the gene selection for the cancer classification problem, according to the different FPGA devices, two processors, and a wide range of values for  $NF$ . The FPGAs provide better speedups than CPUs (up to x9), even for the highest performance processor. We can observe that, the more parallel fitness units we consider, the better speedup we obtain, although this increase is not linear, because of the more dense top level circuits that slow down the clock frequency. In addition, Virtex5 provides better performance than Virtex6 because of the memory constraints to handle the synthesis of large designs (this constraint impedes to consider 256 fitness circuits for the Virtex6 device). Finally, since the Spartan6 device is a low-cost FPGA, it provides much lesser area than the other devices, making it impossible to host more than 32 parallel fitness units.

A similar analysis can be done seeing Fig. 5, that shows the speedups in the biclustering of gene expression data

problem for experiments that use different matrix sizes and parallelizing strategies. Here, we have considered the high and medium-performance FPGA devices and other two different CPUs. Now, we obtain higher speedups than in the former bioinformatics problem (up to x14), and for all the cases, because of the higher parallelization degree in both, the fitness equations and the matrix operations. In addition, we can extract two interesting conclusions. On the one hand, the *MSR fully parallelized* model provides better performance than the *MSR partially parallelized* model for equal bicluster sizes, as the first one involves more parallel operations. Nevertheless, the highest number of replicated floating-point arithmetic operators runs out first the FPGA area available: this is the reason why we can not consider large matrix sizes in the fully parallelized model. On the other hand, when using the *MSR partially parallelized* model, since it parallelizes mainly by rows, we should compare matrix sizes with the same number of rows, for example  $f_{16 \times 8}$  with  $f_{16 \times 16}$ . In this case, we find that the performance is better with fewer columns, as the lower number of floating-point arithmetic operators allows more area to host more fitness units working in parallel, which has more weight in the performance than the bicluster size.

The speedups for the second bioinformatics problem (biclustering) are good in all the cases and higher than for the first problem (gene selection). We find the reason mainly in the parallelism degree of the fitness circuit design, rather than in the number of such circuits working in parallel. The bottom level of the fine-grained parallelization is the fitness circuit, which is composed of some basic floating point operators: adders, dividers,





multipliers and integer to float converters. This way, the more floating point operators running in parallel, the better performance we expect. We find 4 operators in the fitness circuit for gene selection, whereas the fitness implementations for the different bicluster sizes and architectures go from 8 to 32 operators. The number of floating-point operators running in parallel has great influence on the final performance, even more than the number of replicated fitness circuits. In fact, the number of parallel units is higher in the first problem: the performance speedup for the gene selection test with 256 fitness units is  $\times 9$ , whereas 20 units in a  $f8 \times 8$  bicluster gives  $\times 14$ . The reason is simple: a greater number of parallel fitness units in the same FPGA device implies more circuit density in the top level architecture (more communication buses, interconnection blocks, logic cells, etc.), which produces smaller clock frequencies with the corresponding time response decrease.

The ratio of the area occupied by just one fitness circuit to the maximum number of such circuits that the FPGA can host can be seen in Fig. 6, for the second bioinformatics problem: we can have more fitness units in larger FPGAs or considering designs that use lower slice resources. Summarizing, there is a strong relationship between the area required to implement a single fitness function and the bicluster size. Furthermore, increasing the area required for the fitness function decreases the total number of parallel units that can be implemented on FPGA. Therefore, it is needed to establish a tradeoff for each experimental framework.

Finally, it is interesting to know the power consumption of the fitness circuits, since they have an important

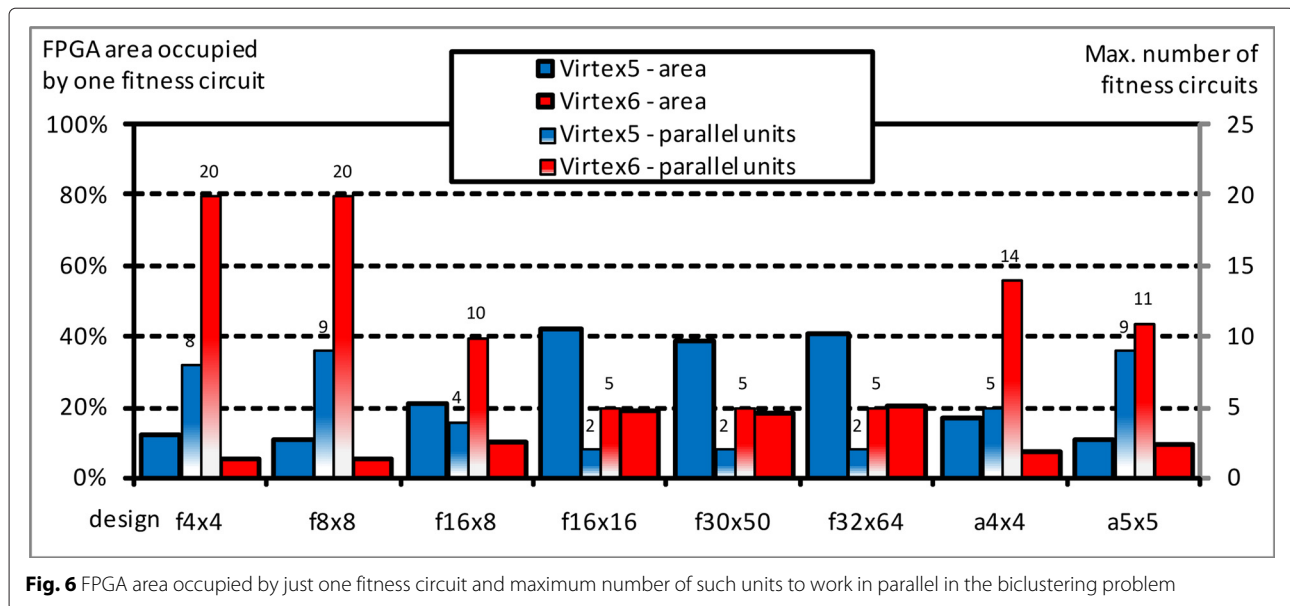
impact in the metaheuristics as we saw in the related work section. This impact involves high energy when the optimization problems demand intensive computations along the time.

The power consumption in the FPGA is calculated by the XPower Analyzer tool inside the Place&Route phase of the implementation, and the CPU energy is measured by the Powerstat tool under Linux, using the Advanced Configuration and Power Interface (ACPI) battery data of a laptop. Considering the gene selection for cancer classification problem with  $NF=8$ , we obtained a power consumptions of 3, 6.4 and 0.2 watts for Virtex5, Virtex6 and Spartan6 devices respectively, whereas the Core2-E6750 processor used around 40 watts for the same configuration. This means that the FPGA reduces the power consumption at least 84 % with regard to the CPU.

## Conclusions

The interest of applying the reconfigurable computing technology based on FPGAs to implement the fitness function lies in the possibility of accelerating the evaluation phase in many metaheuristics. This phase evaluates a population of solutions to a combinatorial optimization problem in the bioinformatics domain. The design of a custom circuit that implements the fitness equation allows its replication in several processing units that work in parallel and, thus, accelerate the evaluation phase.

Since many optimization problems in bioinformatics define fitness functions as floating-point arithmetic operations, we have tested two of them in order to check specific implementation features: area occupation, response time and energy, mainly. From these values we can obtain



the number of replicated units working in parallel and the time for the evaluation phase. The results show that FPGAs provide better performances than CPUs, not only because of the parallelization of the arithmetic operations of the fitness, but also thanks to the concurrent fitness evaluation for several individuals of the population in the metaheuristic.

Finally, the very low power consumption of the FPGA devices in comparison to CPUs proves that FPGA-based parallel computing environments are excellent low-cost computing solutions for intensive computing scenarios.

As future research line, we will tackle the connection of these accelerated fitness functions with evolutionary frameworks for solving the combinatorial optimization problems. The main idea is to implement an EA in software, leaving the intensive fitness computation to the hardware.

## Methods

The methodology for designing and simulating the different circuits considers the software tools described before in Section “Design tools”. Assuming that these tools require depth knowledge in hardware description languages, as well as the corresponding software licenses from the vendors, the general methodology followed in this work is composed of nine steps:

1. Build a hardware project under Xilinx ISE 14.6 environment, selecting the corresponding FPGA device.
2. Design the code files corresponding to the bioinformatics problem: VHDL files. This is the core

step of the work, meaning the greatest effort of the project.

3. Generate the floating-point arithmetic operators from the CoreGen tool.
4. Synthesize and implement the design, activating the corresponding option to obtain advanced reports.
5. After the implementation phase, check the clock period required.
6. Simulate the design using a VHDL testbench, adjusting the clock period to the reported before.
7. Check the time response for the FPGA.
8. Build a C code to run the fitness function in usual microprocessors, compile, run and check the time response.
9. Compare the measured time against the obtained in the FPGA, and calculate the speedup.

## Additional file

**Additional file 1:** This document includes Figures S1, S2, S3 and S4 with detailed views of the top-level and fitness circuits. (PDF 825 kb)

## Funding

This work was partially funded by the Spanish Ministry of Economy and Competitiveness and the ERDF (European Regional Development Fund), under the contract TIN2012-30685 (BIO project: Multiobjective Optimization and Parallelism in Bioinformatics), and by the Government of Extremadura, Spain, with the aid GR15011 to the group TIC015. Ricardo Soto is supported by Grant CONICYT/FONDECYT/REGULAR/1160455 and Broderick Crawford is supported by Grant CONICYT/FONDECYT/REGULAR/1140897.

## Availability of data and materials

The data supporting this research can be found in <http://arco.unex.es/biofpga>. This website contains the source codes and documentation required to implement the hardware accelerators. This repository is freely available for academic purposes.

**Authors' contributions**

JAGP and STA designed the accelerator of the fitness function for the biclustering of gene expression data optimization problem, whereas JLCB did the same for the gene selection for cancer classification problem. In addition, JAGP implemented both bottom-level circuits on FPGA, measuring the speedups with regard to CPUs, whereas JLCB and STA did the same in multi-core CPUs for the biclustering of gene expression data and the gene selection for cancer classification optimization problems, respectively. JMLG supplied experience in modeling fitness functions and solving optimization problems using metaheuristics. RAFD, BC and RS helped to write the article, as well as supply knowledge about metaheuristics. All authors read and approved the final manuscript.

**Competing interests**

The authors declare that they have no competing interests.

**Consent for publication**

Not applicable.

**Ethics approval and consent to participate**

Not applicable.

**Author details**

<sup>1</sup>Department of Technologies of Computers and Communications, University of Extremadura, Polytechnic School, Campus Universitario s/n, 10003 Cáceres, Spain. <sup>2</sup>Department of Computer and Aerospace Engineering, University of Leon, Computer Sciences School, Campus de Vegazana s/n, 24071 Leon, Spain. <sup>3</sup>Pontificia Universidad Católica de Valparaíso, 2362807 Valparaíso, Chile. <sup>4</sup>Universidad Central de Chile, 8370178 Santiago, Chile. <sup>5</sup>Universidad Autónoma de Chile, 7500138 Santiago, Chile. <sup>6</sup>Universidad Espíritu Santo, Guayaquil, Ecuador.

Received: 3 April 2016 Accepted: 24 August 2016

Published online: 31 August 2016

**References**

- Fogel GB, Corne DW. *Evolutionary Computation in Bioinformatics*. Burlington: Morgan Kaufmann; 2003.
- Michalewicz Z, Fogel DB. *How to Solve It: Modern Heuristics*. Berlin: Springer; 2004.
- Ahn CW. *Advances in Evolutionary Algorithms*. Berlin: Springer; 2006.
- Maxfield C. *The Design Warrior's Guide to FPGAs: Devices, Tools and Flows*. Amsterdam: Elsevier; 2004.
- Gokhale M, Graham P. *Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays*. Berlin: Springer; 2005.
- Thomas DB, Howes L, Luk W. 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*. Monterey, CA, USA. New York: ACM; 2009. p. 63–72.
- Che S, Li J, Sheaffer JW, Skadron K, Lach J. Accelerating compute-intensive applications with GPUs and FPGAs. In: *Symposium on Application Specific Processors (SASP 2008)*. Anaheim, California, USA. Washington: IEEE Computer Society; 2008. p. 101–7.
- Segundo EJM, Nedjah N, Mourelle LdM. A scalable parallel reconfigurable hardware architecture for dna matching. *Integr VLSI J*. 2013;46:240–6.
- Fernandez EB, Villarreal J, Lonardi S, Najjar WA. Fhast: Fpga-based acceleration of bowtie in hardware. *IEEE/ACM Trans Comput Biol Bioinforma*. 12;5(2015):973–81.
- Gonzalez-Dominguez J, Wienbrandt L, Kassens JC, Ellinghaus D, Schimmler M, Schmidt B. Parallelizing epistasis detection in gwas on fpga and gpu-accelerated computing systems. *IEEE/ACM Trans Comput Biol Bioinforma*. 2015;12(5):982–94.
- Sukhwani B, Chiu M, Khan MA, Herbordt MC. Effective floating point application on FPGAs: examples from molecular modeling. In: *Proceedings of the High Performance Embedded Computing (HPEC 2009)*: Lexington, MA, USA. Lexington: Massachusetts Institute of Technology; 2009. p. 1–2.
- Chrysos G, Sotiriades E, Rousopoulos C, Dollas A, Papadopoulos A, Kirmizoglou I, Promponas V, Theocharides T, Petihakis G, Lagnel J, Vavylis P, Kotoulas G. Opportunities from the use of FPGAs as platforms for bioinformatics algorithms. In: *IEEE 12th Int. Conference on Bioinformatics and Bioengineering (BIBE 2012)*: Larnaca, Cyprus. Washington: IEEE Computer Society; 2012. p. 559–65.
- Buell D, El-Ghazawi T, Gaj K, Kindratenko V. High-performance reconfigurable computing. *Computer*. 2007;40(3):23–7.
- Sriram V, Leese M. Fpga supercomputing platforms, architectures, and techniques for accelerating computationally complex algorithms. *EURASIP J Embed Syst*. 2009;2009(1):1–2.
- Herbordt MC, Gu Y, VanCourt T, Model J, Sukhwani B, Chiu M. Computing models for fpga-based accelerators. *Comput Sci Eng*. 2008;10(6):35–45.
- Tang W, Yip L. Hardware implementation of genetic algorithms using FPGA. In: *Proceedings of the The 47th Midwest Symposium on Circuits and Systems (MWSCAS '04)*. Washington: IEEE Computer Society; 2004. p. 549–52.
- Peesapati R, Anumandla K, Kudikala S, Sabat SL. Comparative study of system on chip based solution for floating and fixed point differential evolution algorithm. *Swarm Evol Comput*. 2014;19:68–81.
- Rathod A, Thakker RA. FPGA realization of particle swarm optimization algorithm using floating point arithmetic. In: *Proceedings of the 2014 International Conference on High Performance Computing and Applications (ICHPCA)*: Bhubaneswar, India. Washington: IEEE Computer Society; 2014. p. 1–6.
- Omondi AR, Rajapakse JC. *FPGA Implementations of Neural Networks*. Berlin / Heidelberg: Springer; 2006.
- Nedjah N, Mourelle LdM. *Hardware for Soft Computing and Soft Computing for Hardware*. Berlin: Springer; 2014.
- Sidhu RP, Mei A, Prasanna VK. Genetic programming using self-reconfigurable fpgas. In: *Field Programmable Logic and Apps. Lecture Notes in Comp. Science*. Berlin: Springer; 1999. p. 301–12.
- Hidalgo JI, Colmenar J, Risco-Martin J, Sanchez-Lacruz C, Lanchares J, Garnica O, Diaz J. Solving ga-hard problems with EMMRS and GPGPUs. In: *Proceedings of the 2014 Annual Conf. on Genetic and Evol. Computation: Vancouver, Canada*. New York: ACM; 2014. p. 1007–1014.
- Emam H, Ashour MA, Fekry H, Wahdan AM. Introducing an fpga based - genetic algorithms in the applications of blind signals separation. In: *Proceedings of the 3rd IEEE Int. Workshop on System-on-Chip for Real-Time Applications (IWSOC '03)*. Washington: IEEE Computer Society; 2003. p. 123–7.
- Baraglia R, Perego R, Hidalgo JI, Lanchares J, Tirado F. A parallel compact genetic algorithm for multi-fpga partitioning. In: *Ninth Euromicro Workshop on Parallel and Distributed Processing (PDP '01)*. Washington: IEEE Computer Society; 2001. p. 113–20.
- Funie AI, Grigoras P, Burovskiy P, Luk W, Salmon M. Reconfigurable acceleration of fitness evaluation in trading strategies. In: *Proceedings of the 26th IEEE Int. Conf. on Application-specific Systems, Architectures and Processors (ASAP 2015)*: Toronto, Canada. Washington: IEEE Computer Society; 2015. p. 210–217.
- Wang J, Lee CH. *MICAI 2006: Advances in Artificial Intelligence. Lecture Notes in Computer Science*. In: Gelbukh A, Reyes-García CA, editors. Berlin / Heidelberg: Springer; 2006. p. 767–77.
- Zhang Y, Smith SL, Tyrrell AM. Digital circuit design using intrinsic evolvable hardware. In: *Proceedings of the NASA/DoD Conf. on Evolution Hardware (EH '04)*: Seattle, WA, USA. Washington: IEEE Computer Society; 2004. p. 55–62.
- Layzell P. Reducing hardware evolution's dependency on FPGAs. In: *Proceedings of the Seventh International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems (MicroNeuro '99)*: Granada, Spain. Washington: IEEE Computer Society; 1999. p. 171–8.
- Huerta EB, Duval B, Hao JK. A hybrid ga/svm approach for gene selection and classification of microarray data. In: *Lecture Notes in Computer Science*. Berlin: Springer; 2006. p. 34–44.
- Mohamad MS, Omatu S, Deris S, Mismam MF, Yoshioka M. A multi-objective strategy in genetic algorithms for gene selection of gene expression data. *Artif Life Robotics*. 2009;13:410–3.
- Bleuler S, Prelic A, Zitzler E. An ea framework for biclustering of gene expression data. In: *Congress on Evolutionary Computation, 2004 (CEC2004)*. Washington: IEEE Computer Society; 2004. p. 166–73.
- Divina F, Aguilar-Ruiz JS. Biclustering of expression data with evolutionary computation. *IEEE Trans Knowl Data Eng*. 2006;18:590–602.

33. Liu B, Yu C, Wang DZ, Cheung RCC, Yan H. Design exploration of geometric biclustering for microarray data analysis in data mining. *IEEE Trans Parallel Distrib Syst.* 2014;25(10):2540–550.
34. Vasicek Z, Sekanina L. Hardware accelerator of cartesian genetic programming with multiple fitness units. *Comput Inform.* 2010;29:1359–1371.
35. Glette K, Torresen J. A flexible on-chip evolution system implemented on a xilinx virtex-ii pro device. In: *Evolvable Systems: From Biology to Hardware*. Lecture Notes in Computer Science. Berlin: Springer; 2005. p. 66–75.
36. Khabzaoui M, Dhaenens C, Talbi EG. A cooperative genetic algorithm for knowledge discovery in microarray experiments. In: *Parallel Computing for Bioinformatics and Computational Biology*. USA: Wiley; 2006. p. 303–24.
37. Pelta D, Carrascal A. Inverse protein folding on 2d off-lattice model: Initial results and perspectives. In: *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*. Lecture Notes in Computer Science. Berlin: Springer; 2007. p. 207–16.
38. Ambroise C, McLachlan GJ. Selection bias in gene extraction on the basis of microarray gene-expression data. *Proceedings of the National Academy of Sciences of the United States of America.* 2002;99(10):6562–566.
39. Pontes B, Divina F, Giraldez R, Aguilar-Ruiz JS. Virtual error: A new measure for evolutionary biclustering. In: *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*. Lecture Notes in Computer Science. Berlin: Springer; 2007. p. 217–26.
40. Vahid F, Lysecky R. *VHDL for Digital Design*. Hoboken: Wiley; 2007.
41. Pedroni VA. *Circuit Design and Simulation with VHDL*. Cambridge: The MIT Press; 2010.
42. Patterson DA, Hennessy JL. *Computer Organization and Design. The Hardware/Software Interface*. Burlington: Morgan Kaufmann; 2009.

Submit your next manuscript to BioMed Central  
and we will help you at every step:

- We accept pre-submission inquiries
- Our selector tool helps you to find the most relevant journal
- We provide round the clock customer support
- Convenient online submission
- Thorough peer review
- Inclusion in PubMed and all major indexing services
- Maximum visibility for your research

Submit your manuscript at  
[www.biomedcentral.com/submit](http://www.biomedcentral.com/submit)

