Neurocomputing 175 (2016) 899–910

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

A scalable and customizable processor array for implementing cellular genetic algorithms

Martin Letras, Alicia Morales-Reyes *, Rene Cumplido

Instituto Nacional de Astrofisica, Optica y Electronica, Luis Enrique Erro No. 1, Tonantzintla, Puebla, 72840 Mexico

ARTICLE INFO

Article history:
Received 1 December 2014
Received in revised form
12 May 2015
Accepted 12 May 2015
Communicated by Chennai Guest Editor
Available online 6 November 2015

Keywords:
Cellular Genetic Algorithms
Hardware Architecture
FPGA

ABSTRACT

Architectures design for Genetic Algorithms (GAs) has proved its effectiveness to tackle hard real time constrained problems that require an optimization mechanism in one of their phases. Most of these approaches are problem dependent and cannot be easily adapted to other problems. Moreover, GAs based architectures preserve the algorithmic structure of a panmictic population in a sequential GA and therefore they are similar to a software implementation. Recently, combination of GAs, both sequential and parallel and reconfigurable devices such as FPGAs have been merged to create GAs based parallel hardware architectures. This study proposes a novel hardware architectural framework that implements a fine grained or cellular GAs while maintaining toroidal connection among individuals within the population. Achieving massive parallelism is limited by available resources; therefore, the proposed architectural design implements a segmentation strategy that partitions the entire decentralized population while maintaining original algorithmic interaction among solutions. The proposed architecture aims at preserving fine grained GAs algorithmic structure while improving resources usage. It also allows flexibility in terms of population and solutions representation size and the evaluation module containing the objective function is interchangeable.

1. Introduction

Genetic Algorithms (GAs) are metaheuristics inspired in the evolution theory proposed by Darwin. GAs were proposed by Holland [1], and have widely proved to be successful in solving different kind of engineering and scientific problems. They are used as optimization techniques either when a deterministic algorithm is not available or is computationally expensive or when finding an approximate solution to the problem is acceptable. Combinatorial, continuous domain and real-world problems have been successfully tackled by GAs. GAs could be found in applications like robot motion planning [2], digital image processing [3,4], geolocation [5–7] evolvable hardware [5,8,9], etc.

GA are stochastic search techniques, initially they generate a random population of candidate solutions, each solution, also known as individual, is encoding in some form of representation (binary, integer or real numbers) creating a chromosome. Next, genetic operators are applied at solutions representation level in order to explore and exploit the search space. Genetic operators try to mimic the natural selection process in every stage: selection, crossover and mutation. Selection operation, in analogy to natural selection, tries to preserve the fittest individuals from the population. There are different selection criteria like tournament, roulette wheel, etc. Crossover operator tries to emulate the exchange of genetic information in reproduction processes of biological individuals. This operator creates a couple of new individuals called offspring. Each offspring contains part of their parents' genetic material. Crossover operation is useful to explore the space of possible solutions. On the other hand, mutation operation promotes exploitation of solutions at close by regions within the search space. Mutation changes a gene's value in the chromosome according to a mutation probability. A replacement criterion is necessary in order to replace previous individuals in the current population with those evolved. A sequence of these genetic operators is known as a generation. A number of generations is carried out until an approximated solution closed enough to the exact solution is reached.

GAs executes these operations iteratively until an approximated solution closes enough to the exact solution. This process is totally stochastic and it is difficult to know how many generations are needed to converge to an acceptable solution. This could be a disadvantage because there are environments where a response in real time is needed like in embedded systems. A solution is to design dedicated hardware that could be embedded in a system with the purpose of reducing resources usage and of allowing low power consumption. Due to recently advances in FPGA technology, efficient GAs based hardware architectures are implemented using FPGAs as a prototyping tool and later the design could be implemented as an ASIC because the hardware architecture is independent to the employed device.

* Corresponding author.
E-mail address: a.morales@inaoep.mx (A. Morales-Reyes).

http://dx.doi.org/10.1016/j.neucom.2015.05.128
0925-2312/

In this study, a hardware architectural framework is proposed for fine grained or cellular GAs in order to take advantage of their algorithmic massive parallelism together with FPGAs implicit parallelism. The main contribution of this research is to design a partition strategy that is able to segment the decentralized population among a set of processor elements (PEs) while maintaining toroidal connections among individuals. Therefore the original algorithmic structure of a cellular GA is preserved. The proposed architecture can also be configured to support different population and chromosomes sizes and different objective functions can also be plugged. The aim of having this architectural framework is to enable an optimization engine as a functional module in an embedded system.

This paper is organized as follows; in Section 2 related work is discussed as regards previously proposed GAs based hardware architectures both sequential and parallel approaches. Section 3 introduces the algorithmic structure of fine grained or cellular GAs. Section 4 describes the proposed architectural framework for fine-grained GAs continuing in Section 5 with experimental results and a comparison analysis with related works. Finally in Section 6 conclusions and possible future research lines are drawn.

2. Related work

Several hardware architectures haven been proposed to execute GAs, many of these architectures aimed at accelerating the search process and several were specifically designed for accelerating a single processing stage in GAs. Several research works report sequential GAs based designs, and less attention has been paid to fully exploit inner parallelism of implementation platforms such as FPGAs. Moreover, there are few architectural designs that target parallel GAs which in one of their forms are massively parallel at an algorithmic level. Taking advantage of GAs implicit parallelism and implementation platforms such as reconfigurable devices has not been fully explored. Some previously proposed GAs based architectures target designs independent to the problem that offer memory usage reduction because solutions are not represented within. However, there are few proposals of fine grained or cellular GA based architectures designs.

In [25–28], Graphical Processors Units are employed as an alternative to accelerate Cellular Genetic Algorithms. In [25], authors present an implementation of cellular GAs in GPUs to tackle the satisfiability problem 3-SAT, a well-known NP-hard problem. A comparison between GPU and CPU shows a performance improvement for GPU's platform. In [26], PUGACE is introduced as a Cellular Evolutionary Algorithm framework. PUGACE could be configured to work with distinct types of crossover operators, selection operators and distinct fitness function. The framework was tested with the Quadratic Assignment Problem QAP. In [28], an implementation of cellular GAs is carried out, this approach stores individuals and fitness values in the GPU's global memory. Fitness function evaluation and genetic operators are fully implemented in the GPU. Experimental results showed an improvement against CPU implementations. In [27], a multi-GPU implementation of a cellular GA is presented. Several test problems were assessed such as Colville Minimization, Error Correcting Codes Design Problem (ECC) and Massively Multimodal Deceptive Problem (MMDP), and three continuous domain problems, shifted Griewank function, shifted Rastrigin function and shifted Rosenbrock function. Comparison against CPU and single GPU showed that a Multi-GPU implementation overcomes in execution time.

In [11] a GA based hardware architecture using a FPGA Xilinx Virtex2Pro [19] was proposed, population size was defined by 8-bits with a maximum population size of 256 individuals; a flexible number of generations could be defined by 32-bit. Individuals' selection is carried out by roulette-wheel selection and single-point crossover is applied. The fitness function module can be replaced by a different one but the entire designs should be resynthesized. If FPGA resources in one device are not enough to support larger populations or individuals, other FPGA devices can be connected in order to increase overall processing capacity. However, this approach is not fully parallel, neither at an algorithmic level nor at architectural level.

In [12], a compact architecture inspired by the Optimal Individual Monogenetic Algorithm was proposed. This design holds only one individual during algorithm's execution, thus it reduces memory usage in comparison to having a standard population with more individuals. This architecture has two processing stages. One performs a global search generating n individual randomly and holding the fittest; at this stage different regions in the search space are explored. A second stage performs fine changes to the chromosome held, to this purpose; a new genetic operator called micro-mutation was proposed.

An IP core module of a GA to be executed in hardware and that could and could be integrated in an embedded system was proposed in [13]. The main goal was to develop an architecture which is able to use different fitness functions. The architecture contains necessary ports and signals to interchange information from the GA module to the fitness function module. Every time that a different fitness function is used, the new module needs to be loaded and the architecture needs to be resynthesized.

Problem dependency for the objective function module was tackled in [14]. The proposed idea was using Neural Networks (NN) to evaluate the fitness function. Thus, a hardware implementation of a NN within the GA architecture is included. However, neurons weights are calculated in software using Matlab as a tool; after each neuron weight is stored in lookup tables (LUTs). Every time a different fitness function is evaluated, the NN needs to be train again in order to store the new weights; also the architecture design needs to be resynthesized. A sequential GA is implemented together with the NN.

Two GA based hardware architectures were proposed in [15]. One follows the idea of local search while a second one implements a global search criterion. An algorithm called Multiple Different Crossover GA is proposed. This algorithm performs four different crossover operations called: leading, order, annular and DSO crossover. In every generation, the four crossover operators are applied to the parents and the fittest offspring are kept. The algorithmic GA steps of this design are sequential and their proposal heavily depends on the successful application of the four proposed crossover operators.

All previous works are hardware architectural designs that implement sequential GAs that in some cases aim at having flexibility in terms of population size, chromosome size or interchangeable fitness function modules. However recently, fine-grained or cellular GAs have been explored to take advantage of both implicit parallelism at an algorithmic and at processing platform levels. In [21–23], a Compact Cooperative Genetic Algorithm is adapted to work using a Cellular Genetic Structure to tackle evolvable and adaptive hardware to address the scalability issue. In these works, the population is represented as a probability distribution over the set of solutions. At each generation, two individuals are randomly generated from a probability vector. Then, tournament selection is performed over both. Each bit of the probability vector is adjusted according to the result of the tournament selection. Eventually, the cellular GA keeps running until the probability vector has converged.

In [16,17], Dos Santos et al. proposed an architecture that implements a fine-grained GA. A toroidal mesh connection among Processor Elements (PEs) is defined in which each PE has access to two memory

blocks where small subpopulations are stored. According to PEs' toroidal connection, individuals in current subpopulations can be selected more than once for reproduction; thus cellular GAs' canonical algorithmic structure is modified. This architecture executes a coarse-grained parallel GA in which small sub-populations are connected in a toroidal fashion. Memory resources are saved by allocating individuals in this way; however inherent exploration–exploitation ability of cellular GAs is modified. This architecture assessed two combinatorial problems: the Travel Salesman Problem (TSP) and the Spectrum Allocation Problem (SAP).

A cellular compact GA was proposed in [18,24], this architecture is a mixture of a cellular GA and a distributed GA. The architecture maintains toroidal interconnections among individuals like in a cellular GA and within each PE a probability vector is held, it represents the entire population. A compact GA is run based on probabilities stored. The vector is migrated to the nearest neighbors instead of migrating individuals and genetic operations are applied to the probabilities vector instead of to individuals. This architecture applies migration which is an operator applied mainly in coarse-grained or distributed GAs, at the same time the fine grained parallelism is maintained. The objective function aims at classifying signals of electrocardiogram (ECG). A Neural Network is used for signal classification while a GA calculates its weights.

Research works reported in [16–18] aim at memory optimal usage while preserving toroidal connection among PEs. The PEs array in this research proposes a partition strategy to reduce resources usage while preserving at an algorithmic level toroidal connection among individuals taking advantage of full massive parallelism available in this evolutionary technique.

3. Cellular Genetic Algorithm

Sequential GAs use a single population of individuals in panmixia, in this way every individual can mate any other individual of the rest of the population through genetic operations. Solutions independence in GAs makes them suitable for parallel algorithmic approaches. Thus, a rough classification divides them in coarse-grained or distributed and fine-grained or cellular GAs. In cGAs, the population is decentralized and individuals are normally placed on a grid following a toroidal connection among them. It is worth mentioning that several combinations of parallel approaches have been proposed and assessed but due to the application arena of this research, canonical cGAs are approached [10].

Cellular GAs are able to exploit implicit GAs massive parallelism. Fig. 1 shows a square topology of cGAs where each PE corresponds to one individual, thus individuals interact through a local neighborhood, some examples of common neighborhood configuration are shown to the right in Fig. 1; each PE is part of a neighborhood and overlaps other neighborhoods. Because cGAs are massively parallel, solutions are locally exploited and exploration of the search space is carried out globally throughout the entire grid. This is one of the main differences between fine-grained and coarse-grained GAs, not only genetic operators carried out the exploration–exploitation of solutions but also decentralized populations could affect the search process from their topology-neighborhood configuration.

In Algorithm 1, a canonical cGA pseudocode is described. In steps 2 and 4, an initial population is randomly generated and individuals' fitness values are calculated according to the objective function. In step 3, a temporary array (auxiliary pop) is initialized to store evolved individuals within one generation. It is worth mentioning that genetic operators are applied at a local level within neighborhoods, thus individuals that do not belong to the neighborhood cannot affect results of the evolutionary process locally. Steps 5 and 6 define cycles to verify the stop condition and to evolve the whole population synchronously. In step 8, selection operation chooses one of the fittest individuals from the local neighborhood to mate current or central individual. In step 9, two selected chromosomes are recombined using one point crossover, two points' crossover or any other crossover type; crossover promotes exploration within the search space. In step 10, mutation operation is executed with a defined probability which normally is P_m = 1/chrom_length. Mutation carries out small changes at chromosome's genes and therefore further exploitation of solutions takes place. In general, crossover and mutation operations balance the exploration–exploitation trade-off in GAs if no other mechanism to control it is considered. In Steps 11 and 12, children fitness scores are obtained and current individual is replaced by the fittest offspring. New individuals are temporarily stored in auxiliary pop in order to follow a synchronous updating criterion. A sequence of these genetic operations is one generation and a number of generations are executed until the algorithm converges to the solution or fulfills the stop condition (step 5). In step 14, a new evolved population replaces previous one for the next generation corresponding to synchronous updating. In the next section, the proposed processor array with a novel partition strategy to maintain toroidal connection among individuals and therefore cGAs' exploration–exploitation trade-off is described.

Algorithm 1. Cellular Genetic Algorithm.

1. **proc** Evolve(cga)//Parameters of CGA
2. *GenerateInitialPopulation*(cga.pop);
3. auxiliary pop ← cga.pop;
4. *Evaluation*(cga.pop);

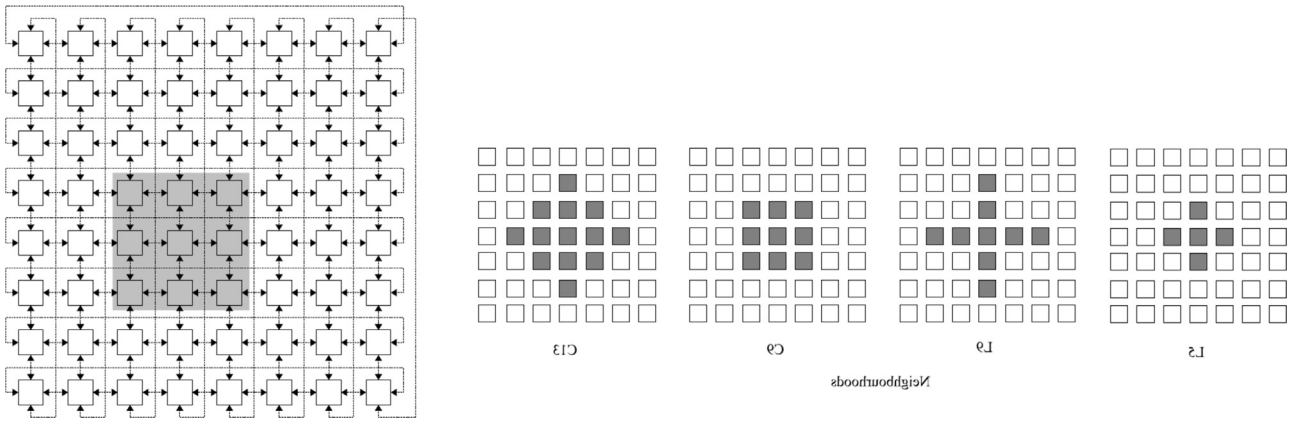


Fig. 1. Processor Elements arrangement in a toroid mesh and the common neighborhood configurations.

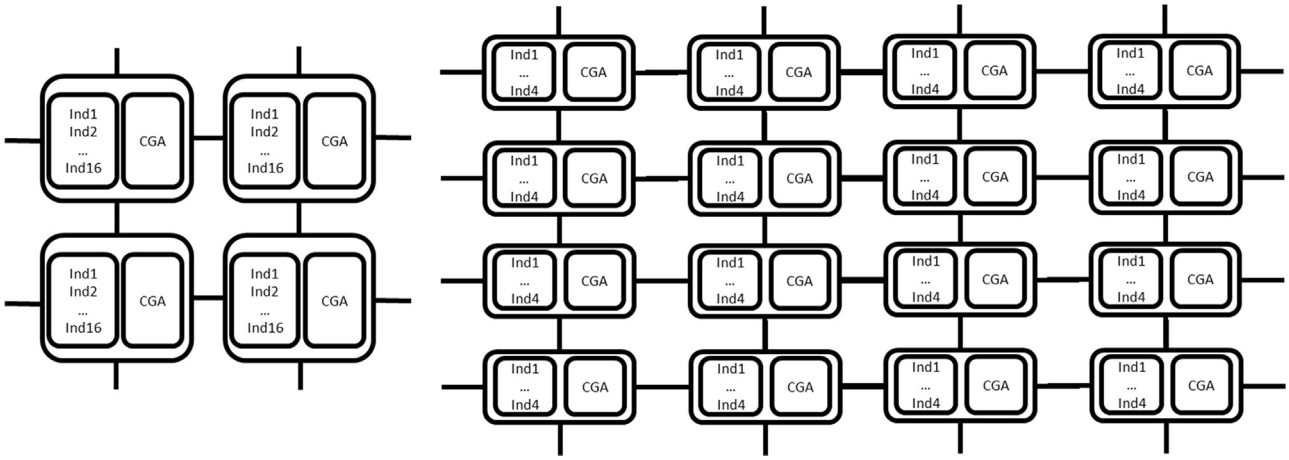


Fig. 2. Two PEs arrangements on a toroidal mesh holding a population of 64 individuals. Left: 4 PE holding 16 individuals each. Right: 16 PE holding 4 individuals each (NB. Border connections are removed for simplicity).

```
5.  while! StopCondition() do
6.    for individual←1 to cga.popSize do
7.      neighbors←GenerateNeighborhood(cga,position
      (individual));
8.      parents←Selection(neighbors);
9.      offspring←Recombination(cga.Pc,parents);
10.      offspring←Mutation(cga.Pm,offspring);
11.      Evaluation(offspring);
12.      Replace(position(individual),auxiliary pop,offspring);
13.    end for
14.    cga.pop←auxiliary pop;
15.  end while
16. end proc Evolve
```

4. Processor array for cellular GAs

In previous sections, the importance of GA and their integration as a part of an embedded system has been explored to justify the importance of a fully parallel design in order to save hardware resources in GA based hardware architectures. An important contribution of this research is to develop a processor array architecture that is able to partition the population's grid of a cGA while maintaining a toroidal connection among individuals and therefore the selective pressure applied due to the population's topology is kept; at the same time at a hardware level the aim is to reuse physical resources. The criterion for population's partition is to divide the whole population among PEs in tiles of different sizes. Each PE process a subpopulation but unlike distributed GAs [16–'18], a logical toroidal mesh is maintained and therefore the inner fine-grain parallelism of this structure is kept.

Scalability in the proposed architecture design is defined at two levels: (1) number of individuals per PE within a squared tile, (2) number of PEs for the overall array. In terms of flexibility the following considerations are needed: for medium size architecture, the processor array would result in neither the fastest nor the most compact design. In contrast, if a compact design is required, more clock cycles would be necessary but hardware resources usage is optimized. However, if time constraints are mandatory, a larger number of PEs can be implemented at a cost of increasing the use of hardware resources. Thus, at first the user should define the processor array size according to specific problem's constraints. The proposed processor array offers flexibility between time constraints and space resources necessary to execute the cGA.

In the next subsection, internal hardware structures designed for the proposed cGA based processor array are described. All sub modules have been developed using a top down strategy at a RTL level. Initially, each module was tested and simulated to verify functionality. Once each module was verified, an integration stage took place at a top level.

4.1. Segmentation strategy

To explain the segmentation strategy the next example is considered: a cGA with 64 individuals (one individual per PE) distributed in a toroidal mesh would exceed available hardware resources because of the internal hardware structures involved in genetic operations and particularly in the fitness function. On the other hand, if only 4 PEs fit within the physical platform, the population can be arranged in such way that each PE holds 16 individuals. Another example considers that hardware resources allow 16 PEs thus the population can be arranged in such way that each PE holds 4 individuals, see Fig. 2. This is an example of 3 different ways for partitioning the decentralized population among PEs. In these scenarios, the same quantity of registers is necessary to store individuals, however it is possible to obtain different occupation area according to how many PEs are available. For example, if the architecture has 4 PEs, the circuitry of one PE is reused to process 16 individuals, and if the architecture has 16 PEs, the circuitry of one PE is reused to process 4 individuals.

In Fig. 2, the left example would require more time to evaluate 16 individuals in every generation but it would reduce hardware resources usage. The right example in Fig. 2 reduces execution time per generation because 16 individuals are evaluated every clock cycle but increases the utilized area because 16 PEs are required. Finding an adequate tradeoff between processing times and hardware resources usage is aimed when hardware architecture is designed to accelerate specific algorithms in this case cellular GAs.

The proposed processor array architecture offers the possibility of selecting one of these scenarios according to the application domain of a top level design. It is worth to remember that the proposed processor array would be attached to an embedded system and would offer certain flexibility. Once the population is distributed among PEs, a strategy to simulate a whole toroidal mesh of 64 individuals is mandatory. For example, if there are 4 PEs with 16 individuals within each PE; wired connections among 64 individuals using only 4 PEs must be emulated; thus the whole grid maintains toroidal connections among individuals, see Fig. 3. Each individual must have a logical connection

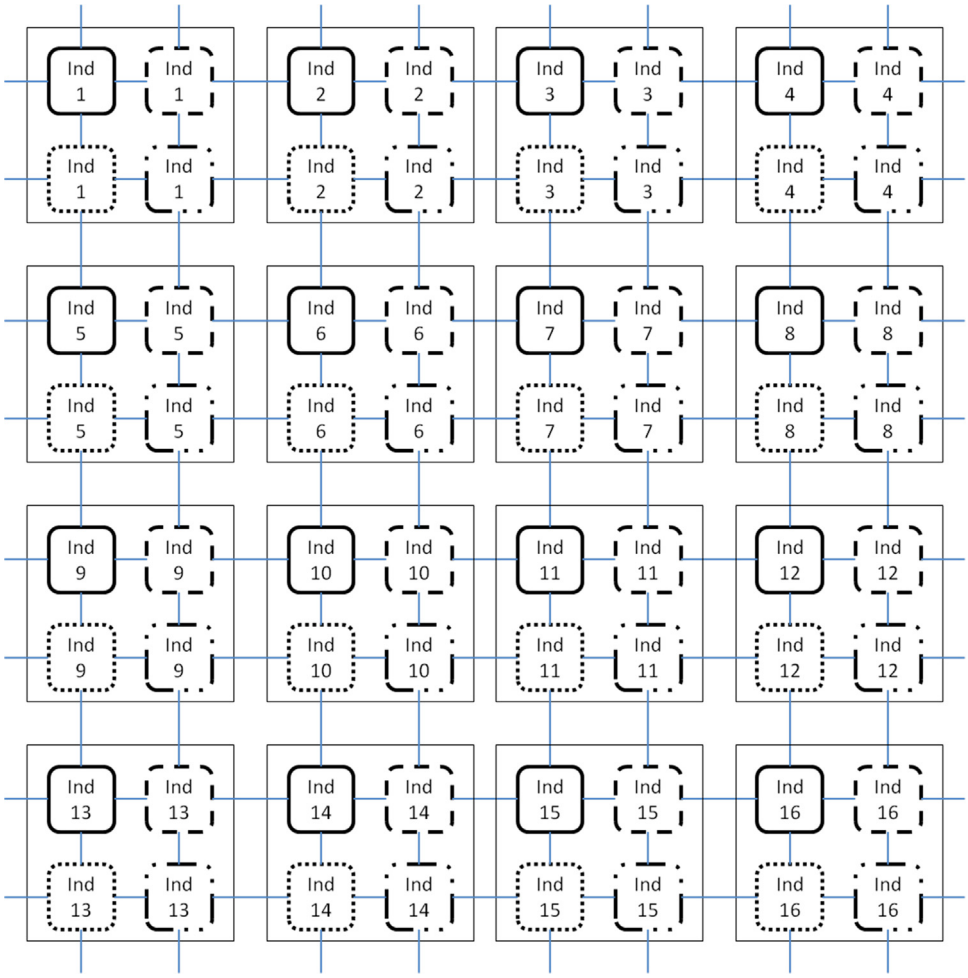


Fig. 3. A logical toroidal mesh simulates connections among 64 individuals using 16 PEs. Solid line squares represent individuals in PE number 1, dotted line squares represent PE number 2 and so on.

with its nearest neighbors, but physically, only connections among 4 PEs exist.

To emulate 64 physically wired PEs array, attention must be paid to information exchanged with neighbor PEs. Fig. 3 shows an example of having 4 PEs each with 16 individuals, thus 16 iterations are needed in order to evolve the whole population. For example, during the first iteration, current individual in each PE is individual number one; solid line squares in Fig. 3. Next, PE number one (corresponding individuals in solid line squares) exchanges current individual and its fitness score to the south and east PEs. However, North neighbor (PE number 3, individuals in round dot line squares) needs individual 5 of PE number 1 (+4 in Hamming distance to current individual), and west neighbor (PE element number 2, individuals in dash line squares) needs information of individual number 2 in PE number 1 (+1 in Hamming distance to current individual). After modeling information behavior of what is necessary to exchange with neighbors, it was observed that inner PEs only need to exchange current individual with its neighborhood. It was also observed that PEs in the frontier needs to send individuals different to current one. Therefore, Algorithms 2 and 3 are proposed to deal with these scenarios.

In Algorithms 2 and 3, variables i and j represent PE's position within the processor array. For example, indexes [0, 0] correspond to PE number one. Variable *DIM* defines the number of PEs per row, with a total of 4 PEs, $DIM=2$. *Increment* variable is calculated by dividing number of columns in the logical toroidal mesh between *DIM*, in the example, an increment equal to 4 is obtained. *Individuals* variable is the number of individuals per PE, 16 for this example. These parameters are needed for information exchange to the closest neighbor. In Algorithm 2 example: $north \leftarrow ind5$ because $index \leftarrow ind\_act + increment$ where $ind\_act = 1$ and $increment = 4$, thus $index = 5$ and individual 5 is sent to the North output; the other output in Algorithm 2 is $south \leftarrow ind\_act$ then individual 1 is sent to the South output, thus individual 5 and individual 1 are received by PE number 3.

Algorithm 2. Exchanging information with North and South neighbors.

```
1. proc South_North_information(j,DIM,ind_act,increment,
   individuals)
2. north ← ind_act;
3. south ← ind_act;
4. if i == 0 then
5.       index ← ind_act + increment;
6.       if index ≥ individuals then
7.          index ← index-individuals;
8.       end if;
9.       north ← index;
10.      elsif i == DIM-1 then
11.       index ← ind_act-increment;
12.       if index ≥ individuals then
13.          index ← index + individuals;
14.       end if;
15.       south ← index;
16.      end if;
```

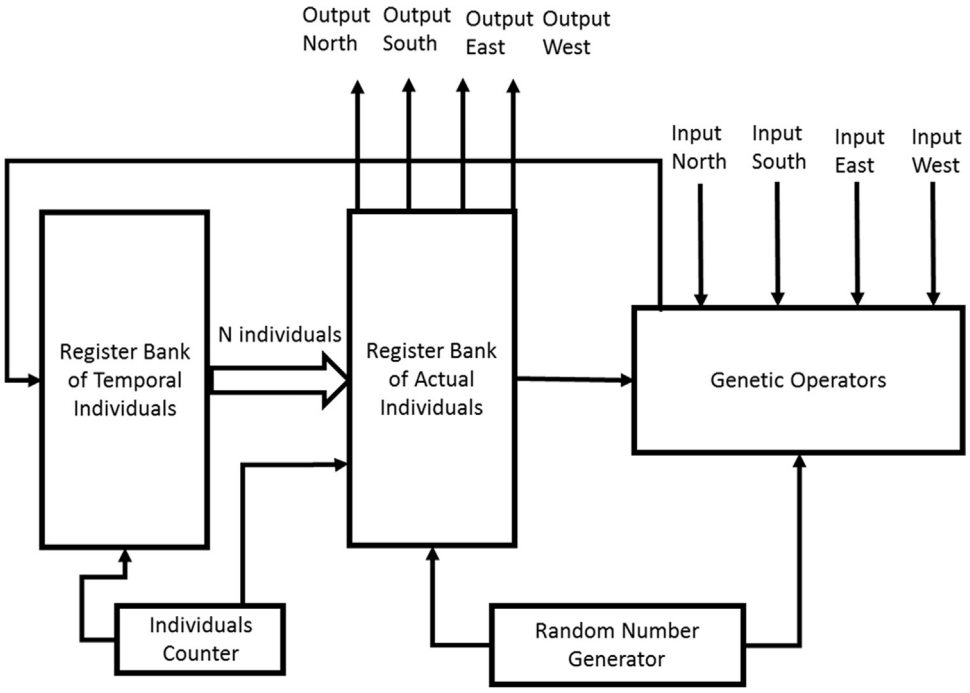


Fig. 4. Internal structure of the proposed PE.

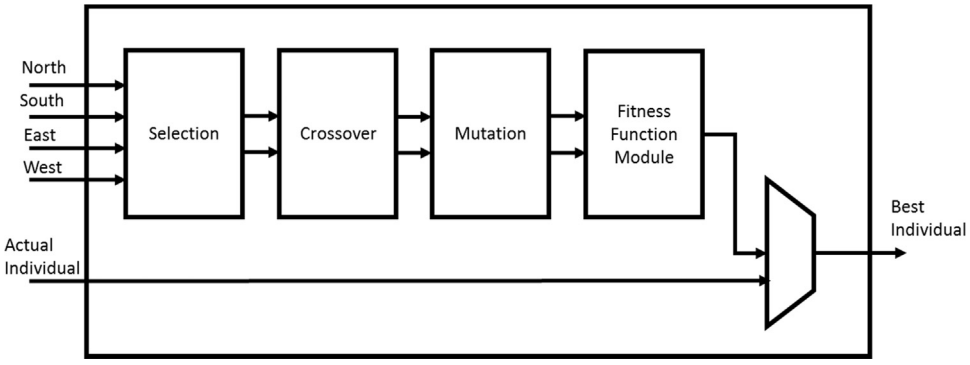


Fig. 5. Internal structure of genetic operator module.

In Algorithm 3, first output is *west*←*ind*2 because *index*←*ind_act*+1 where *ind_act* = 1, thus *index* = 2 and individual 2 is sent through West output; second output in Algorithm 3 is *east*←*ind_act* then individual 1 is sent through East output; individuals 1 and 3 are received by PE number 2. Algorithms 2 and 3 guarantee that PEs send and receive corresponding algorithmic data. The proposed control mechanism is implemented within registers bank of actual individuals, see Fig. 4. This means that each PE needs to control current individual and neighbors in and out.

Algorithm 3. Exchanging information with West and East neighbors.

```
1. proc West_east_Information(j,DIM,ind_act,increment,
   individuals)
2. west←ind_act;
3.    east←ind_act;
4. if j== 0 then
5.       index←ind_act+1;
6.       if index > increment*(floor(ind_act/inc_ver)+1) then
7.          index←increment*(floor(ind_act/inc_ver)+1)-
   increment;
8.       end if;
9.       west←index;
10.      elsif j==DIM-1 then
11.   index←ind_act-1;
12.      if index < increment*floor(ind_act/inc_ver) then
13.         index←increment*floor(ind_act/inc_ver);
14.      end if;
15.      east←index;
16.      end if;
```

4.2. Processor Element

A PE has two registers banks, a pseudorandom number generator, a counter and the genetic operations module, see Fig. 4. Each PE is also a systolic processor because an initial seed is propagated for each PE on the fly, it also has the option of changing the initial seed for a PE; it also exits the best individual

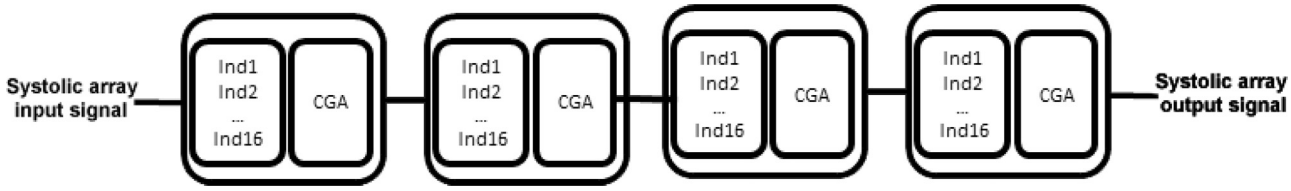


Fig. 6. Systolic processor arrangement in a row.

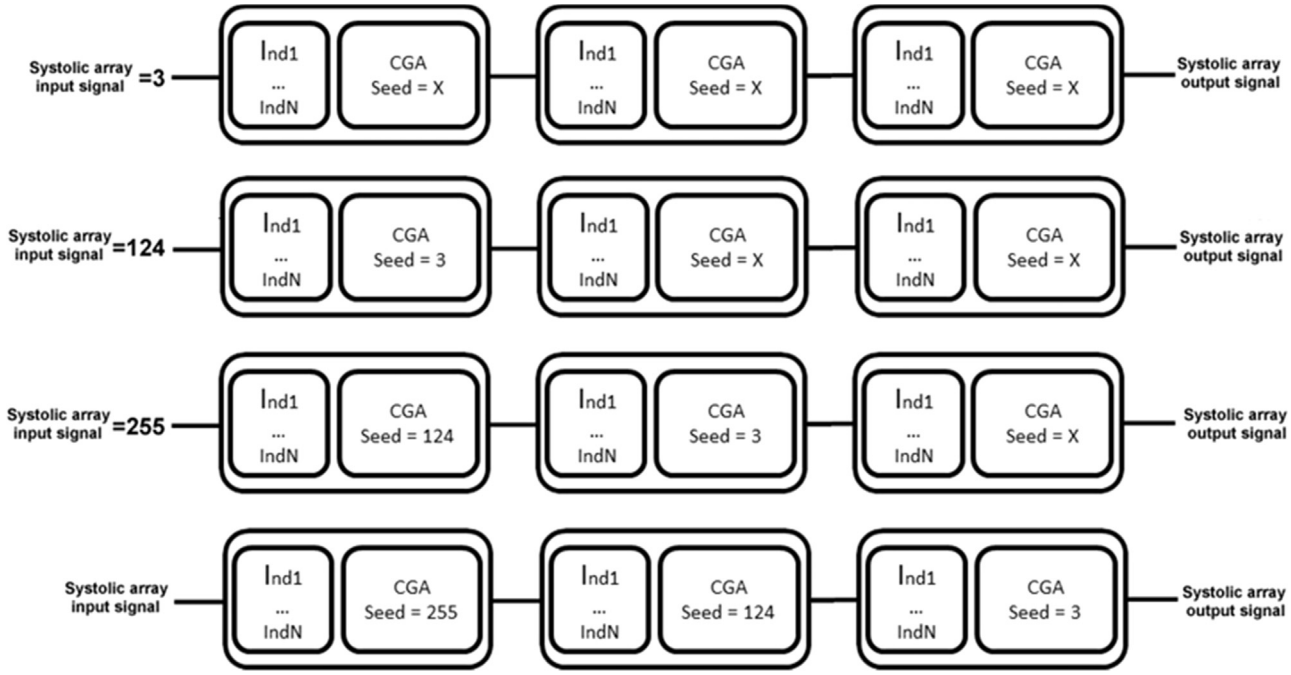


Fig. 7. Initializing each PE with a different seed using systolic array signals.

chromosome when the cellular GA has converged to the problem's solution. Fig. 6 draws the systolic array's structure.

Two banks of registers are defined, one to store current individuals, and another for temporary individuals storage. After each PE receives its seed, the register bank for current individuals is loaded randomly. Next, the processor array evaluates individuals and carries out the evolutionary process through genetic operations, new individuals are stored in the temporal bank register for synchronous cGA's updating. Once a generation is completed, chromosomes pass from temporary to a current individuals' registers bank. Individuals at a temporary registers bank are necessary because information cannot be erased from the registers bank for current individuals because another PE could require their information. A counter indicates current individual and when a complete generation is assessed.

Fig. 5 shows the internal structure of the genetic operations module. Once individuals arrive to a PE, a selection process for the best individual among them, considering current individuals, is carried out. Then recombination and mutation are applied to the selected parents. Finally, offspring are evaluated and the genetic operator module outputs the fittest individual. In following subsections more details about every internal module of the genetic operations module are provided.

4.2.1. Pseudo random number generator

Previously, it has been mentioned that GAs are stochastic processes because almost at every stage, a random parameter is required, either when recombining parents or mutating children. For this module, a cellular automata array to generate pseudo random numbers is used. After a careful review of the literature, cellular automata guarantees good quality of random numbers sequences; this approach is used in [11,13,14]. In this study, a combination of rules 90 and 150 is used, for more details readers are referred to [20].

Each PE has a Pseudo-Random Number Generator (PRNG). In order to initialize cGA's solutions and configuration parameters, the systolic array signals are used. Each row in the processor mesh receives a seed at the systolic array input signal and one clock cycle after, a new seed is received while previous one is sent to the next PE. In Fig. 7 an example is drawn: in the first clock cycle, the systolic array input signal has a 3 value. In the next clock cycle, the first PE has received value 3 as a seed. One clock cycle after, the first PE has received value 124 and has sent value 3 to the second PE. In the next clock cycle, the first PE has received value 255 and has sent value 124, while the second PE has received value 3. This process is repeated until all PEs have received corresponding seeds. The systolic array has been chosen because it is an efficient way to propagate information among a set of PEs.

4.2.2. Selection module

The internal structure of the selection module is shown left in Fig. 8. Binary tournament selection, one of the most common selection methods used in GAs, is implemented. Binary tournament selects two fittest individuals from the neighborhood. This operator receives as inputs north, south, east and west individuals and their fitness function value. This operation is implemented using comparators connected in cascade; see left in Fig. 8.

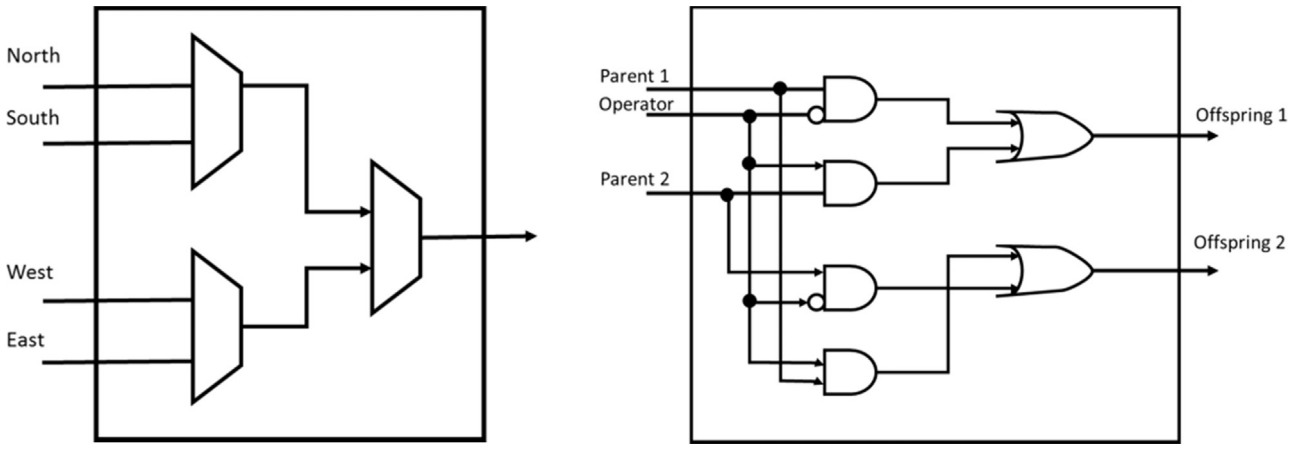


Fig. 8. Left: tournament selection operator. Right: crossover operator.

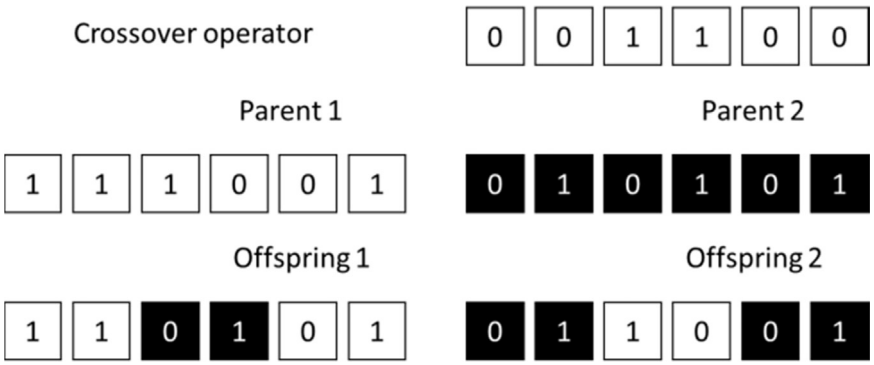


Fig. 9. Crossover module's behavior. Parents genetic information is inherited to the offspring.

4.2.3. Crossover operation's module

In order to simplify the crossover operation, its design is divided in two phases. If, the behavior of the crossover module is observed, the operator could be implemented using only *or* and *and* logical gates. The procedure employed is illustrated in Fig. 9. There are two strings for chromosome parents and one that represents the crossover operator. This crossover string indicates the exchange position of the genetic material. In Fig. 9 example: "001100" is the string representing the crossover operation. Zeros in the crossover string indicates that the new chromosome remains same as its parents, ones in the crossover string indicates chromosome's sections are interchaged between parents. Designing in this way the crossover operator, it is simplified and recombination is performed by only using *and* and *or* gates as it is shown at the right in Fig. 8. The overall function of this module consist in reading 2 random numbers from the PRNG module and filling with 1 all positions within the interval of these random numbers.

4.3. Mutation operation's module

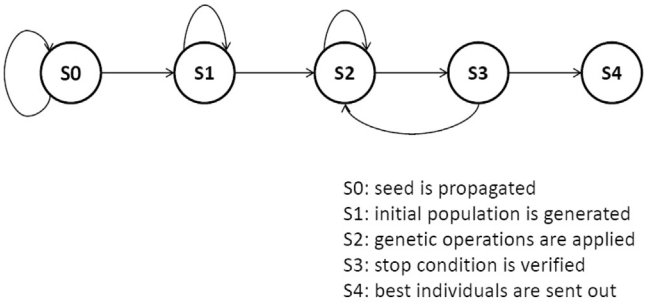
Mutation is performed by chromosome's bit flipping according to a mutation probability. At an architectural level, it is necessary to generate a random mutation probability per bit which is a demanding task. Therefore, a n random number generator for log (n) bits was implemented, where n is the chromosome size; thus a n*log (n) length binary string is calculated as a mutation probability vector. Mutation probability per chromosome's gene is calculated by $P_{mutation} = \frac{1}{2^{\log(n)}}$; if a zero is found, corresponding gene position within the chromosome is flipped.

4.4. Control mechanism

Fig. 10 illustrates the control mechanism proposed in this study. A five state Finite State Machine (FSM) is defined aiming at a fully parallel architecture. In *S0*, a Pseudo-Random Number (PRN) acting as a seed is propagated throughout PEs using systolic array ports as shown in Fig. 6. The FSM stays in this state until all PEs have a different seed. In *S1*, each PE generates a new individual in one clock cycle and stores its chromosome in the register bank of actual individuals. The FSM machine remains in this state n clock cycles, where n corresponds to the number of individuals of the corresponding population's segmented tile. Once the initial population is generated, the FSM move to state *S2*, in this state, the genetic operations module is executed. The number of clock cycles spent by the processor array in one generation can be calculated as follows:

$$T_{generation} = n_{individuals} * n_{cyclesfitness}$$

where $n_{individuals}$ corresponds to the number of individuals in the corresponding population's segmented tile, $n_{cyclesfitness}$ is the number of clock cycles used to calculate the fitness function. Timing to calculate selection, crossover and mutation is not considered because these modules were designed using combinational logic. However, it is important to know how many clock cycles are required in a generation because it could be relatively easy to calculate in how much time the whole processor array would converge to a problem's solution. Once a generation is evaluated, the FSM advances to *S3*; in this state the stop condition is verified. If the cGA has assessed a predefined number of generations, the FSM advances to the next state; if this limit has not been reached, the FSM returns to *S2*. In the final state *S4*, the



$$T_{generation} = n_{individuals} * n_{cyclesfitness}$$

Fig. 10. Finite state machine used as control mechanism of each PE.

architecture has to send out the best individual from individuals evaluated by every PE. The number of cycles necessary at this state is equal to n, where n corresponds to the number of PE in a row of the toroidal grid.

5. Results analysis

The proposed processor array for cellular GAs is simulated and synthesized in a Zynq XC7Z020 Xilinx FPGA with -1 grade speed using VHDL as programming language [19]. Xilinx ISE is used for design and synthesis process. The hardware architecture is simulated at RTL in Xilinx ISIM simulator. Standard benchmark problems in the evolutionary computation arena are used as benchmark problems. Three combinatorial problems are implemented: ISO PEAK, MAX ONE and MMDP, in order to evaluate architectural performance in terms of latency and resources usage required by the proposed processing framework. Because the assessed problems are combinatorial, only once clock cycle is required to calculate individual's fitness value. The main objective of this empirical assessment is to demonstrate that using the proposed segmentation strategy allows to balance hardware resources usage and the number of clock cycles required by the cGA to converge to a problem's solution. It is worth to remember that, the canonical structure of a cellular GA is not modified by the proposed partition strategy and that individuals maintain their toroidal connections during evolution. Benchmark fitness functions are defined in the next subsection.

5.1. Benchmark problems

Three benchmark problems were implemented in order to evaluate the proposed processor array that implements a canonical cGA. In this research, combinatorial problems were assessed because their fitness calculation requires one clock cycle for execution. In this way, the overall performance of the processor array can be assessed as a framework for implementing cGAs independently from the objective function but having the possibility of interchangeable modules to tackle other optimization problems.

5.1.1. Massively Multimodal Deceptive Problem (MMDP)

MMDP is a problem composed by q sub-problems. The fitness value of each sub-problem reflects the number of ones (unitation) each sub-problem has. A very simple lookup table with assigned values is used, see Table 1. The number of local and global optima would depend on the size of the problem. In this research, a size of $q=6$ sub-problems has been used. Therefore the fitness function will sum up individual fitness per sub-problem (x) and a value of $q=6$ will be obtained when the global optimum is reached. The

Table 1
MMDP lookup table.

Number of ones	Sub function value
0	1.00000
1	0.00000
2	0.36038
3	0.64057
4	0.36038
5	0.00000
6	1.00000

Table 2
ISO PEAK fitness function.

$\vec{x}$	*00*	*01*	*10*	*11*
Iso1	m	0	0	$m-1$
Iso2	0	0	0	m

fitness function is given by:

$$F_{MMDP}(\vec{x}) = \sum_{i=1}^{q} fitness_{x_i}$$

where $fitness_{x_i}$ is calculated using Table 1.

In Table 1, values indicate that each sub-problem has a deceptive point in the middle and two global maxima at the extremes. This problem presents a large number of local optima in comparison to the number of global ones which is $2q$, where q is the number of sub-problems.

5.1.2. MAX ONE

This problem consists of maximizing the number of 1 s in a chromosome. The maximum fitness function value is k, where k is the length of the binary string. A problem size with $k=64$ has been defined for experimental purposes.

5.1.3. ISO-PEAK

ISO-PEAK is a non-separable problem which means its variables affect each other at a genetic level modifying solutions' fitness scores. In this study, each individual is encoded in a binary vector with length n, where $n = 2 \times m$ (a chromosome is divided in two groups). For experimental purposes $n=64$ bits thus $m=32$. This fitness function is defined in Table 2 based on the following equation:

$$function_{ISOPEAK} = Iso2(x_1, x_2) + \sum_{i=2}^{m} Iso1(x_{2i-1}, x_{2i})$$

5.2. Experimental results

In order to evaluate the proposed processor array, previously reported approaches are included in Table 3. However, a direct comparison between the proposed approach and other architectures' proposals is not feasible; not only different devices were used for implementation but also different limits to configure algorithmic parameters were considered in terms of population size, chromosomes length, selection criteria, crossover and mutation operators, stop conditions, etc.

Table 3 draws a summary of closely related works as a reference for the proposed hardware architecture. One of the main objectives in these studies is to accelerate algorithmic convergence to the solution. All of them obtained good results when compared to software implementations but few compared their performance results with other hardware implementations. In [16], an array of

Table 3
Hardware architectures related work.

Work	Max. pop size	Max. ind. length (bit)	Selection	Crossover operator	Device employed	Frequency of operation (MHz)	Time for 10^6 generations (s)	Registers	LUTS	Slices	BRAM
[11]	65,535	16	RW	1-Point	Virtex II Pro	50	–	–	–	–	–
[12]	256	8	Tournament	2-Point	–	300	–	–	–	–	–
[13]	–	–	Tournament	2 Point	Altera APEX 20 k	30	–	–	–	–	–
[14]	–	16	RW	2 Point	Spartan 3	12.5	–	–	–	–	–
[15]	65,535	16	RW	Different Crossover	Virtex 4	85	–	462	10,153	5489	–
[15]	65,535	16	RW	Different Crossover	Virtex 6	399	–	6498	222	5616	–
[16]	128	150	Tournament	1-Point	Virtex 6	179	3.10	2020	2605	913	12
[16]	128	150	Tournament	1-Point	Virtex 6	152	1.38	7485	9306	3727	48
[16]	128	150	Tournament	1-Point	Virtex 6	122	0.78	27,591	34,885	13,316	192
[17]	192	~1024	Tournament	1-Point	Virtex 6	179	3.10	35,848	36,266	15,782	49
[17]	192	~ 1024	Tournament	1-Point	Virtex 6	152	1.38	47,092	51,664	21,949	61
[17]	200	~ 1024	Tournament	1-Point	Virtex 6	122	0.78	62,807	72,653	31,262	77
[18]	256	–	Tournament	2-Point	Virtex 5	280	–	1642	5506		

PEs in a toroidal mesh is used to implement a distributed GA. Three different array sizes were implemented: 2×2, 4×4, 8×8 PEs aimed at solving the Travel Salesman problem (TSP). An extension to this work is presented in [17], where the Spectrum Allocation Problem is tackled. Approaches presented in [16–18] are parallel GAs hardware architectures that define toroidal connections among PEs; therefore these are considered as reference for the proposed cellular GA based architecture.

In this study, for each fitness function, three different processor arrays'configurations were synthesized for 64 individuals as the population size in all cases: (1) an array of 4×4 PEs with a partition's grid of 2×2 individuals, (2) an array of 2×2 PEs with a partition's grid of 4×4 individuals, and (3) an array of 8×8 PEs with a partition's grid of 1×1 individual. Every individual has a maximum length of 64 bits, except individuals for the MMDP where a chromosome length of 66-bit is required. Having an array of PEs of 1×1 is not considered in this research, because this configuration corresponds to a sequential GA with panmictic population, and therefore the evolutionary cycle cannot be parallelized. This architecture was designed as a sub-module for a system on top. Then, the wrapper for this architecture must be implemented according to the needs of a top level system. Necessary inputs are seeds, clock and reset signals. For experimental purposes, a 64-bit seed port could be used but in order to avoid excessive use of input blocks, only a 1-bit signal is used thus the seed is feed serially. To output the cellular GA result, only one output of an individual's length is necessary. Once the cellular GA reaches a maximum number of generations, the systolic array starts to take out individuals, thus each row in the mesh is connected to a FIFO to store all the individuals and then take them out serially.

For comparison purposes, the proposed cGA architecture is also evaluated in software using a PC with an Intel i3-3217u at 1.80 GHz processor with 8 GB of RAM memory and in alternate hardware platform using and Embedded Processor Dual Core ARM Cortex A-9 with 1 GHz CPU frequency and 512 MB RAM memory on a Zynq 7020 SoC. In both cases, ANSI C is used for compilation. Soft version of the cellular GA emulates parallelism at an algorithmic level but implements a sequential version for execution preserving original toroidal connection among individuals with in cGA's population. Processing time results are presented in Table 4 clearly showing the advantage of designing a parallel architecture for cellular GAs.

Tables 5, 6 and 7 show performance metrics to evaluate each benchmark problem. Resources usage or area information includes the number of Flip Flop registers, Look up Tables and slices used

Table 4
Cellular GA results on an Intel i3 GPP and an ARM processor. Population size: 64 individuals, chromosome size: 64 bits.

Problem	Time for 10^6 generations with i3 processor (s)	Time for 10^6 generations with i3 processor (min)	Time for 10^6 generations with ARM processor (s)	Time for 10^6 generations with ARM processor (min)
MAX-ONE	263.574	4.392	1288.519	21.475
ISO-PEAK	241.510	4.025	1255.401	20.923
MMDP	344.388	5.739	1924.538	32.075

for each testing case. Operational frequency and number of clock cycles per generations are used to calculate the overall time it takes for the architecture to execute a certain number of generations. It is worth to remember that clock utilization of each processor array configuration could be different according to each testing problem. Each problem defines a different data path and clock resources reported here are after synthesis results.

The processor array that implements 8×8 PEs is the fastest one, however it also consumes the largest area and hardware resources and specifically for the used device the proposed architecture surpasses its size, see Table 8 for Zynq 7020 available resources.. On the other hand, the architecture implementing a 4×4 PEs array is slower than having a processor array of 64 PEs with 1 individual evolving per PE, but it optimizes space in area and hardware resources usage. Finally, the architecture that only uses 4 PEs in a 2×2 processor array is slower than the previous cases; but it is the most compact because only uses 4 PEs to iterate through the grid's partitions. In all cases, the algorithmic structure of a cellular GA is preserved and therefore the algorithmic performance is the same for the three assessed configurations. The main difference relies on how hardware resources are used and how this affects the overall performance in terms of the number of clock cycles.

An initial comparison among the proposed processor array against an i3 processor and an ARM processor reveals that in some cases a speed improvement of 3 orders of magnitude is achieved. This occurs because the i3 processor is a General Purpose Processor and several functions are sequentially executed while in the proposed architecture a high level of parallelism is performed. A similar situation is presented for the ARM processor; the hardware architecture achieved in some cases 4 orders of magnitude speedup, this is because to the limited capacity of the embedded processor. These results provide a stronger experimental support for the proposed cGA based hardware architecture showing

Table 5
Clock cycles and area required by the ISO PEAK problem.

Processor array	Registers	LUTs	Slices	LUT FF pairs	Clock cycles per generation	Frequency of operation (Mhz)	Time for 10^6 generations (s)
2×2	24,716	15,717	23,244	7458	17	46.240	0.36764
4×4	55,664	33,400	36,970	13,704	5	46.240	0.10813
8×8	177,536	103,945	78,014	21,467	2	46.240	0.04420

Table 6
Clock Cycles and area required by the MMDP problem.

Processor array	Registers	LUTs	Slices	LUT FF pairs	Clock cycles per generation	Frequency of operation (MHz)	Time for 10^6 generations (s)
2×2	25,428	14,998	23,981	8106	17	45.284	0.33150
4×4	56,628	28,897	38,203	15,570	5	47.522	0.10521
8×8	182,748	86,184	80,186	28,840	2	47.436	0.04216

Table 7
Clock cycles and area required by the MAX ONE problem.

Processor array	Registers	LUTs	Slices	LUT FF pairs	Clock cycles per generation	Frequency of operation (MHz)	Time for 10^6 generations (s)
2×2	24,716	22,276	23,392	7442	17	26.208	0.64865
4×4	55,664	52,501	36,986	13,579	5	26.362	0.18966
8×8	177,536	181,842	78,078	21,072	2	26.359	0.07587

Table 8
Zynq 7020 total of resources available.

	Z-7020
Programmable logic	Artix 7
No. of slices	85,000
No. of flip flops	106,400
No. of 6 input LUTs	53,200
No. of 36 Kb block RAMs	140
No. of DSP48 slices	220

significant advantages of having a dedicated hardware architecture framework for implementing cellular GAs versus software and therefore sequential approaches.

Although it is not possible to carry out a direct comparison to other proposed approaches either at an algorithmic level or at an implementation platform level, the closest related works that are reported in [16, 17]. The architecture's design presented in [16] reports a lower consumption of hardware resources, however the proposed architecture does not require using BRAMs blocks. In contrast, comparing the proposed architecture with the architectural design reported in [17], better resources usage is achieved in this research while maintaining the advantage of not requiring BRAM blocks. On the other hand, the proposed architecture reports lower operational frequencies but it requires a lower number of clock cycles per generation. A possible reason is that the processor array proposed in this article integrates a PRNG in every PE while their design uses one PRNG which is shared among PEs; therefore more time is needed to propagate random numbers which are required during GAs evolution.

6. Conclusions and future work

In this paper, a novel processor array for the implementation of fine-grained or cellular Genetic Algorithms has been developed. The main contribution of this research is the segmentation strategy to partition a decentralized population among an array of processor elements. This strategy provides flexibility for different application domains and their requirements. For example, the processor array could be configured as a compact relatively slower or as a faster and resource demanding optimization engine, the final decision depends on the requirements of the embedded system. Together with this architectural design, an algorithmic characteristic is preserved which differentiates cellular GAs from other parallel genetic algorithmic approaches: toroidal connection among individuals is maintained and therefore balancing the exploration–exploitation trade-off from a structural perspective is preserved. This is achieved by using logical addressing to emulate a physical wired architecture. For future work, it is possible to improve the architectural design in order to reduce the occupied area and to increase the operational frequency using advanced pipelining techniques. Moreover, specific selection criteria of cellular GAs can be considered within the proposed design in order to modify the selective pressure applied during the search. Moreover, achieving dynamic configuration of several algorithmic parameters would imply a flexible control of the exploration–exploitation trade-off. From a topology perspective dimension is another structural characteristic that could be explored to further improve diversity during the search.

Acknowledgments

Martin Letras is supported by the Mexican National Council for Science and Technology (CONACyT), scholarship number 298024.

References

[1] John H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, The MIT Press, 1992, ISBN: 9780262581110.
[2] Yanrong Hu, Simon X. Yang, A knowledge based genetic algorithm for path planning of a mobile robot, in: Proceedings of IEEE International Conference on Robotics and Automation, ICRA'04, vol. 5, no., 26 April–1 May 2004, pp. 4350–4355, doi: 10.1109/ROBOT.2004.1302402.
[3] B.C.H. Turton, T. Arslan, An architecture for enhancing image processing via parallel genetic algorithms and data compression, in: Proceedings of the First International Conference on Genetic Algorithms in Engineering Systems:

Innovations and Applications, GALESIA. (Conf. Publ. No. 414), 12–14 September 1995, pp. 337–342, doi: 10.1049/cp:19951072.
[4] Sara Hashemi, Soheila Kiani, Navid Noroozi, Mohsen Ebrahimi Moghaddam, An image contrast enhancement method based on genetic algorithm, Pattern Recognit. Lett. 31 (13) (2010) 1816–1824, http://dx.doi.org/10.1016/j.patrec.2009.12.006 1 October.
[5] Xu Jiangning, T. Arslan, Wang Qing, Wan Dejun, An EHW architecture for real-time GPS attitude determination based on parallel genetic algorithm, in: Proceedings of NASA/DoD Conference on Evolvable Hardware, pp.133–141, 2002, doi: 10.1109/EH.2002.1029877.
[6] Xu Jiangning, T. Arslan, Wan Dejun, Wang Qing, GPS attitude determination using a genetic algorithm, in: Proceedings of the 2002 Congress on Evolutionary Computation, CEC'02, 12–17 May 2002, pp. 998–1002, doi: 10.1109/CEC.2002.1007061.
[7] E.F. Stefatos, T. Arslan, High-performance adaptive GPS attitude determination VLSI architecture, in: IEEE Workshop on Signal Processing Systems, SIPS 2004, 13–15 October 2004, pp. 233–238 doi: 10.1109/SIPS.2004.1363055.
[8] J.R. Evans, T. Arslan, The implementation of an evolvable hardware system for real time image registration on a system-on-chip platform, in: Proceedings of NASA/DoD Conference on Evolvable Hardware, pp. 142–146, 2002, doi: 10.1109/EH.2002.1029878.
[9] E.F. Stefatos, T. Arslan, An efficient fault-tolerant VLSI architecture using parallel evolvable hardware technology, in: Proceedings of 2004 NASA/DoD Conference on Evolvable Hardware, 26–26 June 2004, pp. 97–103, doi: 10.1109/EH.2004.1310816.
[10] E. Alba, B. Dorronsoro, Cellular genetic algorithms. Computational Science & Engineering, Springer, 2008, ISBN: 978-0-387-77610-1.
[11] P.R. Fernando, S. Katkoori, D. Keymeulen, R. Zebulum, A. Stoica, Customizable FPGA IP core implementation of a general-purpose genetic algorithm engine, IEEE Trans. Evol. Comput. 14 (1) (2010) 133–149, http://dx.doi.org/10.1109/TEVC.2009.2025032.
[12] Z. Zhu, D.J. Mulvaney, V.A. Chouliaras, Hardware implementation of a novel genetic algorithm, Neurocomputing 71 (1–3) (2007) 95–106, http://dx.doi.org/10.1016/j.neucom.2006.11.031, ISSN: 0925-2312.
[13] Chen Pei-Yin, Chen Ren-Der, Chang Yu-Pin, Shieh Leang-san, H.A. Malki, Hardware implementation for a genetic algorithm, IEEE Trans. Instrum. Meas. 57 (4) (2008) 699–705, http://dx.doi.org/10.1109/TIM.2007.913807.
[14] Nedjah Nadia, Mourelle Luiza de Macedo, An efficient problem-independent hardware implementation of genetic algorithms, Neurocomput 71 (1–3) (2007) 88–94, http://dx.doi.org/10.1016/j.neucom.2006.11.032.
[15] R. Faraji, H.R. Naji, An efficient crossover architecture for hardware parallel implementation of genetic algorithm, Neurocomputing 128 (2014) 316–327.
[16] P.V. Dos Santos, J.C. Alves, J.C. Ferreira, A Scalable Array for Cellular Genetic Algorithms: TSP as Case Study, ReConFig, 2012, pp. 1–6.
[17] P.V. Dos Santos, J.C. Alves, J.C. Ferreira, A framework for hardware cellular genetic algorithms: an application to spectrum allocation in cognitive radio, in: Proceedings of 23rd International Conference on Field Programmable Logic and Applications (FPL), 2013, pp. 1–4.
[18] Y. Jewajinda, P. Chongstitvatana, A parallel genetic algorithm for adaptive hardware and its application to ECG signal classification, Neural Comput. Appl. 22 (7–8) (2013) 1609–1626.
[19] All Programmable Technologies from Xilinx Incorporation. ⟨http://www.xilinx.com⟩ (last visited: October, 2014).
[20] P.D. Hortensius, R.D. McLeod, Werner Pries, D.M. Miller, H.C Card, Cellular automata-based pseudorandom number generators for built-in self-test, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 8 (8) (1989) 842–859, http://dx.doi.org/10.1109/43.31545.
[21] Y. Jewajinda, P. Chongstitvatana, FPGA implementation of a cellular compact genetic algorithm, in: Proceedings of IEEE NASA/ESA Conference on Adaptive Hardware and Systems A,HS'08, 2008. pp. 385–390.
[22] Y. Jewajinda, P. Chongstitvatana, Cellular compact genetic algorithm for evolvable hardware, in: Proceedings of IEEE 5th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, ECTI-CON 2008, 2008, vol. 1, pp. 1–4.
[23] Y. Jewajinda, An adaptive hardware classifier in FPGA based-on a cellular compact genetic algorithm and block- based neural network, in: Proceedings of IEEE International Symposium onCommunications and Information Technologies, ISCIT 2008, 2008 pp. 658–663.
[24] Y. Jewajinda, P. Chongstitvatana, FPGA-based online-learning using parallel genetic algorithm and neural network for ECG signal classification, in: Proceedings of 2010 IEEE International Conference on Electrical Engineering/Electronics Computer Telecommunications and Information Technology (ECTI-CON), pp. 1050–1054.
[25] Z. Luo, H. Liu, Cellular genetic algorithms and local search for 3-SAT problem on graphic hardware, in: Proceedings of IEEE Congress on Evolutionary Computation, CEC 2006, 2006, pp. 2988–2992.
[26] N. Soca, J.L. Blengio, M. Pedemonte, P. Ezzatti, PUGACE, a cellular evolutionary algorithm framework on GPUs, in: Proceedings of 2010 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8.
[27] P. Vidal, E. Alba, A multi-GPU implementation of a cellular genetic algorithm, in: Proceedings of IEEE Congress on Evolutionary Computation (CEC), 2010, pp. 1–7.
[28] P. Vidal, E. Alba, Cellular genetic algorithm on graphic processing units, in: Proceedings of Nature Inspired Cooperative Strategies for Optimization, NICSO 2010, Springer, Berlin, Heidelberg, 2010, pp. 223–232.

Martin Letras received the B.Sc. degree in Computer Science from the Autonomous University of Puebla, Mexico, in 2013. Currently, he is a currently working towards his M.Sc. degree at the Computer Science Department from the National Institute for Astrophysics, Optics and Electronics, Mexico. His research interests are algorithmic acceleration via hardware architectures and embedded systems design.

Alicia Morales Reyes was admitted to the Ph.D. degree in the College of Science and Engineering at the University of Edinburgh in 2011, UK. She received the M.Sc. degree in Computer Science (INAOE) in 2006 and the B. Eng. degree in Electrical and Electronics Engineering (UNAM) in 2002, Mexico. She is an associate researcher at the Computer Science Department in INAOE. Among her research interests are the improvement of evolutionary algorithmic techniques and the design of hardware architectures inspired on biological principles for algorithmic acceleration while tackling problems in different application areas such as optimization, machine learning, signals and imaging processing.

Rene Cumplido received the B.Eng. from the Instituto Tecnologico de Queretaro, Mexico, in 1995. He received the M.Sc. degree from CINVESTAV Guadalajara, Mexico, in 1997 and the Ph.D. degree from Loughborough Univerity, UK in 2001. Since 2002 he is a professor at the Computer Science Department at INAOE in Puebla, Mexico. His research interests include the use of FPGA technologies, custom architectures and reconfigurable computing applications. He is co-founder and Chair of the ReConFig international conference and founder editor-in-chief of the International Journal of Reconfigurable Computing. He also serves as associate editor of several international journals.