Fast and compact evolvable systolic arrays on dynamically reconfigurable FPGAs

Javier Mora; Andrés Otero; Eduardo de la Torre; Teresa Riesgo

***Abstract*—*Evolvable hardware* may be considered as the result of a design methodology that employs an *evolutionary algorithm* to find an optimal solution to a given problem in the form of a digital circuit.**

Evolutionary algorithms typically require testing thousands of candidate solutions, taking long time to complete. It would be desirable to reduce this time to a few seconds for applications that require a fast adaptation to a problem. Also, it is important to consider architectures that may operate at high clock speeds in order to reach very speed-demanding situations.

This paper presents an implementation on an FPGA of an evolvable hardware image filter based on a *systolic array* architecture that uses *dynamic partial reconfiguration* in order to change between different candidate solutions. The neighbor to neighbor connections of the array offer improved performance versus other approaches, like Cartesian Genetic Programming derived circuits. Time savings due to faster evaluation compensate the slower reconfiguration time compared with virtual reconfiguration approaches, but, at any rate, reconfiguration time has been improved also by reducing the elements to reconfigure to just the LUT contents of the configurable blocks.

The techniques presented in this paper lead to circuits that may operate at up to 500 MHz (in a Virtex-5), filtering 500 megapixels per second, the processing element size of the array is reduced to 2 CLBs, and over 80 000 evaluations per second in a multiple-array structure in an FPGA permit to obtain good quality filters in around 3 seconds of evolution time.

***Index Terms*—FPGA, evolvable hardware, systolic array, partial reconfiguration, LUT**

I. Introduction

Evolvable hardware (EH) is the result of a design methodology that allows obtaining hardware solutions to problems not known in depth by using an *evolutionary algorithm* (EA) to test different solutions until finding one that fulfills the requirements of the given problem. Moreover, it provides a way to make hardware adapt to tasks and requirements that change over time, allowing the same hardware to be reused.

EAs usually require evaluating several thousands or even millions of candidate solutions to find a satisfactory one. This *training* stage can take minutes or even hours to complete, during which the system will not be operative or will be working at a sub-optimal configuration. In order to minimize this training time, it is important to speed up the evaluation process as much as possible.

Although it is possible to perform evaluations by simulation, it is often faster to perform them directly on the target hardware. This is possible with FPGAs, which allow changing the digital circuit they implement via reconfiguration rather than implementing a fixed one. Moreover, some FPGAs can perform *partial reconfiguration*, which permits the incremental modification of the circuit in a similar manner to how some EAs refine a solution by making incremental changes on them and evaluating the newly obtained solution.

Certain FPGAs are able to do this from within the FPGA itself, allowing such partially reconfigurable systems to be implemented autonomously on the FPGA, without requiring an external agent to run the EA and manage the reconfiguration. This is known as *dynamic partial reconfiguration* (DPR).

The training time of an implementation of EH based on DPR will be determined by these times:

- The **reconfiguration time** in which a candidate solution is set up.
- The **evaluation time** in which the candidate solution is executed in order to measure its performance and conformance to the requirements.
- The **time overhead** involved in the EA, which is usually performed in software. Most of this time will involve generating a new candidate solution.

For simple EAs, the latter will be small and can be neglected compared to the other two.

This paper shows an implementation that presents very good reconfiguration times and processing speeds, thus incrementing the number of candidate solutions evaluated per second and reducing drastically the evolution time.

The rest of the paper is structured as follows: Section II introduces the state of the art and possible alternatives. Section III describes the chosen solution and implementation details. Section IV evaluates the implementation and lists the results obtained from it. Finally, section V shows the conclusions.

II. Technical background and previous work

Common EH-based processing systems consist of a large number of basic processing units, known as *processing*

elements (PEs), which are interconnected in a specific manner. Each of these PEs has a certain number of inputs coming from the system input or from other PEs, implements a specific operation on the data it receives from these inputs, and sends the processed result to other PEs, typically registering the result in order to create a pipelined data processing architecture. The mission of the EA is to determine which operation will be performed by each PE and how the PEs will be interconnected; these parameters constitute a specific *candidate solution*.

A. Interconnection topologies

Given that allowing every PE to get its inputs from any other possible PE in the system would lead to excessively complex routing (which is generally bad for FPGA design) and to having excessively big multiplexers at the inputs of the PEs, the way in which PEs can interconnect is usually restricted so that only a few possible interconnections are allowed.

One of these interconnection topologies is the *cartesian genetic programming* (CGP) [1], which consists of a series of PEs arranged in columns, as seen in Fig. 1. Each of these PEs can take data from the inputs and the columns to the left, and usually implements a stateless simple function such as arithmetic addition or logic AND. In order to further simplify the system in terms of multiplexers and routing, the number of inputs available to a certain PE can be constrained to a maximum number of columns to the left (typically one column, to avoid large multiplexers).

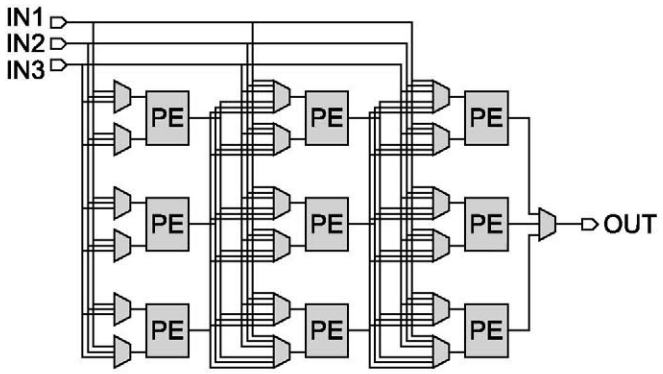

Fig. 1. Example of a 3×3 CGP topology with 3 inputs and 1 output. Each PE in this example has 1 output and 2 inputs, from either the system input or a PE in the previous column.

The main problems with this topology are that the routing of the nets may still be too complex, and that the multiplexers introduce extra delay in the logic path reducing the frequency of operation, although this can be solved by registering the output of the multiplexers in addition to the PE outputs.

Multiplexers also have the drawback of using a high amount of resources. For example, while an 8-bit adder processing element only needs 2 slices in total (1/4 per bit, corresponding to a single LUT) in modern Xilinx FPGAs, a single 13:1 multiplexer as proposed in [2] (9 inputs + 4 PEs) requires 1 slice per output bit [3], 16 slices in total for 2 input multiplexers. Therefore, multiplexers alone would represent an 89% of the resource usage for this topology.

Another topology which does not suffer this problem is the *systolic array*, first defined in [4], as a generic computing engine, and used as a reconfigurable fabric for implementing EH in [5]. It was originally intended for complex PE operations but it can be used with simpler PEs as well. Opposite to CGP, inputs to each PE are fixed, connecting each of them to its neighbors (Fig. 2).

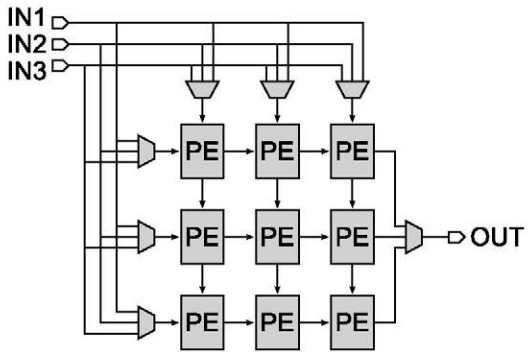

Fig. 2. Example of a 3×3 systolic array with 3 inputs and 1 output. Notice how the routing has been simplified and the multiplexers removed except for the ones at the input and output of the system.

This topology simplifies the routing between processing elements, allowing for shorter data paths and thus lower delay. Also, smaller logic resources per every PE permits PEs to be closer between neighbors. Additionally, it removes the need of having a multiplexer at the input of every PE (except at the system input), thus again reducing the delay as well as the resource usage. Its simplicity is also an advantage in dynamically scalable systems such as [6].

The disadvantage of this solution is the degradation of connectivity, which would force EAs to take longer evolution cycles to obtain correct mappings.

B. Reconfiguration methods

As said before, PEs must be able to switch between different functions according to the configuration of a specific solution.

The most straightforward way to achieve this is by simply implementing all possible circuits and using a multiplexer to select which function the PE uses (Fig. 3), in a similar way to how an ALU works. This is known as *virtual reconfigurable circuit* (VRC), and has the advantage of being highly portable (independent of the FPGA used). However, this approach is considerably resource and energy consuming, since all the possible functions have to be implemented at once, and the extra multiplexer to select the used function has the same problems CGP has: it introduces extra delay and resource usage.

This approach is used in [7] in combination with CGP in order to implement an evolvable image filter.

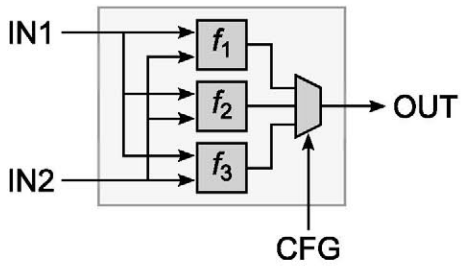

Fig. 3. VRC with 3 functions. Output may be registered (not shown).

An alternative to VRCs consists in using DPR if the FPGA supports it. With this methodology, rather than having to implement all possible PE circuits physically in the FPGA, only one of them is present at a time. A *reconfiguration engine* will later be able to replace this circuit with another one stored in a PE library by partially reconfiguring the area of the FPGA corresponding to the PE. (Fig. 4)

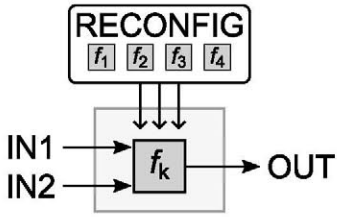

Fig. 4. By using DPR instead of VRCs, the output multiplexer can be removed and only one function is implemented, reducing the amount of resources used.

This approach reduces considerably the amount of resources used by a single PE and potentially improves its maximum frequency of operation, at the expense of making the system specific to a certain FPGA model. Additionally, DPR introduces a reconfiguration time overhead, unlike VRCs where this time is negligible since it would only involve changing the input of some multiplexers; however, in [8] the reconfiguration time is still small compared to the evaluation time.

The design of modules interchangeable by means of partial reconfiguration requires the input and output nets of the module to be compatible with the ones on the static system. Xilinx tools provide a solution for this requirement in their *partial reconfiguration flow* [9]. However, this flow does not support using the same circuit on different positions even though this is theoretically possible due to the uniformity of FPGAs.

Bus macros present an alternative to Xilinx's partial reconfiguration flow, and are used in in [5] in conjunction with a systolic array topology. A bus macro is a circuit with fixed routing used for anchoring nets, making the inputs and outputs of partial circuits compatible. As a downside, bus macros introduce extra delay and resource usage.

Custom tools such as *Dreams* [10] or *GoAhead* [11] allow anchoring certain nets through specific routes without needing bus macros. Additionally, they ensure that no other nets cross the reconfigurable area boundaries. The Dreams tool is used in [6] to implement a scalable systolic array of up to 8×7 PEs.

A third way to change the functionality of a circuit is to change the content of the FPGA LUTs that are used to implement logic functions, which is not as flexible as changing the complete circuit including interconnection nets, but can still be a good solution if the PEs are similar, and is easier to accomplish given that no special routing considerations have to be taken. This was done in [12] on a Xilinx Virtex-II Pro by temporarily using the LUTs as shift registers in order to write a new content. However, the amount of LUTs that can work as shift registers has decreased in more modern Xilinx FPGA families, and currently only 25–30% of the LUTs in Virtex-5 [13] and 7 series [3] FPGAs have this functionality, unlike Virtex-II Pro which allowed this in all of its LUTs [14].

[2] uses a similar LUT reconfiguration approach in a CGP topology, but using DPR instead of relying on a shift register functionality, thus being able to use all the LUTs in the FPGA, not only the aforementioned 25–30% of them.

III. Implementation

The chosen implementation is based on a *systolic array* with a size of 8×8 PEs that uses *dynamic partial reconfiguration of lookup tables* for changing the functionality of both PEs and multiplexers given its speed and ease of implementation.

Each PE has two 8-bit inputs, fixed to its *north* and *west* PEs respectively, and transmits its 8-bit output to both the *east* and *south* PEs, as was shown in Fig. 2. At array level, data are fed through the north and west sides, and the output is selected from the east outputs, leaving the south ones unused.

The target platform is a Xilinx Virtex-5 LX110T FPGA. One advantage of this FPGA family over more modern alternatives such as the 7 series families is that its minimum reconfiguration unit (*frame*) is a 60% smaller [15], [16], thus reducing reconfiguration times for small areas.

The reconfiguration engine has been implemented as a custom hardware peripheral that is able to write one word per clock cycle to the *internal configuration access port* (ICAP) of the FPGA. It differs from the one described in [8] in that it has been simplified and specifically adapted to fine grain reconfiguration. Xilinx's XPS HWICAP [17] is not used because its speed is constrained by single word transactions on the SoPC bus, resulting in very low reconfiguration speeds.

A. System description

The application of this implementation is a noise image filter based on a 3×3 pixel sliding window. This filter will be used to filter grayscale images of 128×128 8-bit pixels at a speed of 1 pixel per clock cycle.

The EH system is implemented as a MicroBlaze peripheral with the described systolic array; 3 BRAM memories of 16 KB each, for storing the 128×128 pixel input, output, and reference images; and a comparator that calculates the difference between output and reference as the *sum of absolute*

errors (SAE), which is obtained pixel by pixel as

$$SAE = \sum_i \sum_j |a_{ij} - b_{ij}|$$

where a and b are the output and reference images, and i and j the pixel coordinates.

Both the memories and the comparator have a throughput of 4 pixels per clock cycle, and are connected to the systolic array through dual-clock asymmetric FIFOs so that they can operate at different clock speeds and do not become the bottleneck of the system.

In order to improve the evolution time and make a better use of the FPGA area, 8 systolic arrays and 8 comparators have been implemented that can filter the same image in parallel, as was done in [18] and [2] with 3 and 6 filters respectively. Only the output of one of the arrays is stored; the rest are only used during the training stage for evaluating a specific solution.

B. Processing element description

PEs are built by directly instantiating lookup tables, following an approach based on the one used in [2]. Each PE is implemented with only 2 Virtex-5 CLBs, using 2 LUTs and 1 FF per bit and dedicated carry logic, as shown in Fig. 5.

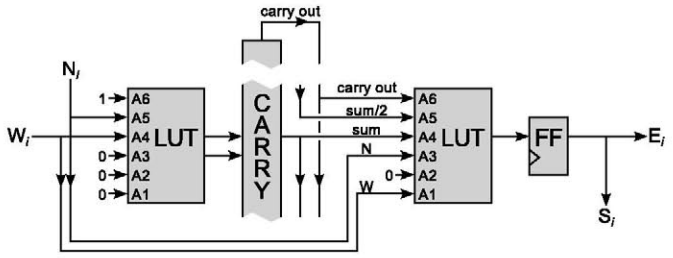

Fig. 5. Fragment of a PE. Each PE is constituted by 8 fragments like this, 1 per bit. The *sum/2* signal is the *sum* signal (extended with the *carry out* bit) shifted 1 bit to the right.

PEs are composed of 2 stages, one for calculating a sum and carry derived from the inputs and one for generating a result based on the inputs, sum, and carry. The first stage can also be used for multiplying one input by 2 (by adding it to itself) or comparing the inputs.

The choice of LUT inputs is not arbitrary and is made in order to minimize the circuit delay and reduce the amount of FPGA *frames* that need to be reconfigured to 1 frame for the first stage LUT and 2 frames for the second one, rather than the 4 frames needed to reconfigure each LUT completely [2].

This configuration allows a broad range of processing functions to be implemented. The current implementation uses the 16 functions used in [5], which are listed in table I, all of which can be implemented in the described circuit.

C. Input and output selectors

Input and output selectors of the systolic array are also implemented using reconfigurable LUTs rather than actual multiplexers, so they are switched by reconfiguring the FPGA instead of changing an input signal.

TABLE I
FUNCTIONS IMPLEMENTED BY THE PES

Index	Equation	Index	Equation
0	$N+W \bmod 256$	8	$\lfloor \frac{N}{2} \rfloor$
1	$2 \cdot N \bmod 256$	9	$\lfloor \frac{W}{2} \rfloor$
2	$2 \cdot W \bmod 256$	10	N
3	$\min(N+W,\ 255)$	11	W
4	$\min(2 \cdot N,\ 255)$	12	$\max(N, W)$
5	$\min(2 \cdot W,\ 255)$	13	$\min(N, W)$
6	$\lfloor \frac{N+W}{2} \rfloor$	14	$\max(N-W,\ 0)$
7	255	15	$\max(W-N,\ 0)$

Rather than implementing a 9:1 multiplexer, each input selector uses a 2-stage model similar to that of PEs, shown in Fig. 6. The first stage selects a row of the 3×3 pixel window, and the second one adds a certain amount of latency in order to shift the pixel to the left, since the window moves 1 pixel per clock cycle to the right. A similar approach has been previously done in [6].

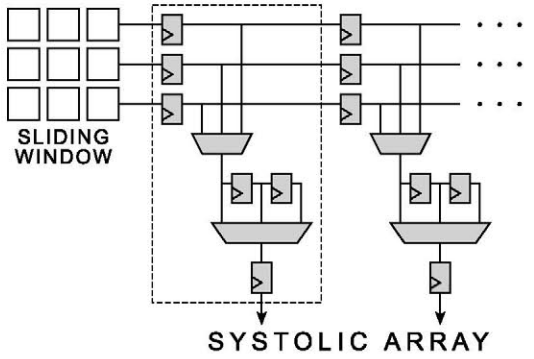

Fig. 6. Diagram of an input selector (dotted) showing its cascaded connection. Each multiplexer symbol in the picture is implemented using 8 reconfigurable LUTs, one per bit.

The output selector is implemented as shown in Fig. 7, using one reconfigurable LUT per bit per systolic array output.

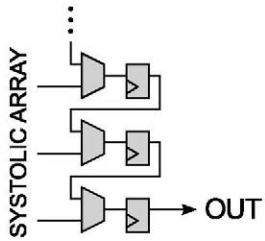

Fig. 7. Diagram of the output selector.

Input and output selectors are cascaded, delaying the data 1 clock cycle in every stage. This not only helps breaking long lines into shorter ones to avoid timing problems, but

also corrects the latency of data propagation, making it independent of the path the data followed inside the systolic array. This makes the latency of the systolic array constant and predictable.

D. Evolutionary algorithm

The chosen EA is a simple mutation-only *genetic algorithm* (GA) running in software. Each gene of the GA represents either the function of a PE or the configuration of an input or output selector. The GA has a population of a single parent and an offspring of a single child ((1+1)-EA) and a mutation rate of 2; this is, in each generation a new candidate is generated by modifying 2 randomly chosen genes of the parent, evaluating the resulting candidate, and substituting the parent with the new candidate in case it has a better or equal performance, or *fitness* value.

The random genes to modify are chosen so that all modifications in a single mutation take place in the same column of the systolic array. This is done because reconfiguration affects a whole column due to the nature of Xilinx FPGAs, so reconfiguring multiple elements on a single column will take as long as reconfiguring a single one, thus saving reconfiguration time. A similar strategy is used in [2], to a level of single frames rather than complete PE columns.

The fitness criterion used is the SAE between a noise-free reference image and the result of filtering a noisy one, with lower values representing better solutions. (The resulting filter will not be specific to the training image and can be used with other images with similar noise type and levels, as shown in [5].)

In order to take advantage of the multiple parallel arrays, 8 independent evolutions are run in parallel, one in each systolic array. At the end of the evolution, the best of the 8 results is chosen. Having multiple short evolutions is usually a better strategy than having a single long one, since it reduces the risk of getting stuck at a sub-optimal local minimum. Additionally, in order to promote good evolutions and discard bad ones, every 2048 generations the best evolution is forked and the worst one is terminated.

Such strategy was already used in [19] to implement a distributed EA on a network of intercommunicated nodes capable of running independent evolutions in each, to keep diversity, but exchanging candidates to give preference to best populations.

IV. Results

A. Hardware characteristics

The described system has been implemented in a Xilinx Virtex-5 LX110T FPGA with a speed grade of -1. The resource distribution is shown on Fig. 8. The 8 systolic arrays use 2688 slices (16% of the available FPGA slices), 336 per array; and the rest of the system (MicroBlaze soft processor, reconfiguration engine, and logic for controlling the systolic arrays) 3433 slices (20%).

Fig. 8. Resource distribution of the system in a Virtex-5 LX110T FPGA, with the 8 systolic arrays visible on the top right quadrant.

The systolic arrays work at a speed of 400 MHz (400 megapixels per second, which is equivalent to 24 000 images of 128×128 px per second), the reconfiguration engine at 200 MHz, and the MicroBlaze at 100 MHz. However, a system with up to two 16×16 PE systolic arrays has been tested with speeds of up to 500 MHz, showing that the system is scalable and that the frequency bottleneck is probably not in the systolic array but in the controlling logic.

Each PE uses 2 vertically arranged CLBs (4 slices), making a column of up to 10 PEs fit in a single Virtex-5 clock region, although a systolic array may span multiple clock regions. The systolic arrays are compactly implemented, using 100% of the slices in the local area, as shown in Fig. 9. This promotes short nets and therefore improves timing.

B. Experimental results

Table II shows the average SAE values obtained after 100 independent runs of the EA for a *Lena* image with *salt and pepper* noise levels of 5%, 10%, and 20% (Fig. 10), comparing the result of the current parallelized (1+1)-EA with the single-threaded (1+8)-EA used in [5], both taking advantage of the 8 systolic arrays to accelerate the evolution. Both algorithms are run for 32 768 generations, evaluating a total of 262 144 candidate solutions, although intermediate results after 65 536 (1/4 of the evolution time) and 131 072 evaluations (1/2) are also shown.

Fig. 10 also shows some of the results obtained with the filters resulting from the parallelized (1+1)-EA.

As can be seen, the results achieved using the former EA are comparable to similar approaches [2], [5]. Nevertheless, the current EA achieves better results even with 4 times fewer evaluations (and in a time 5 times shorter). The final SAE

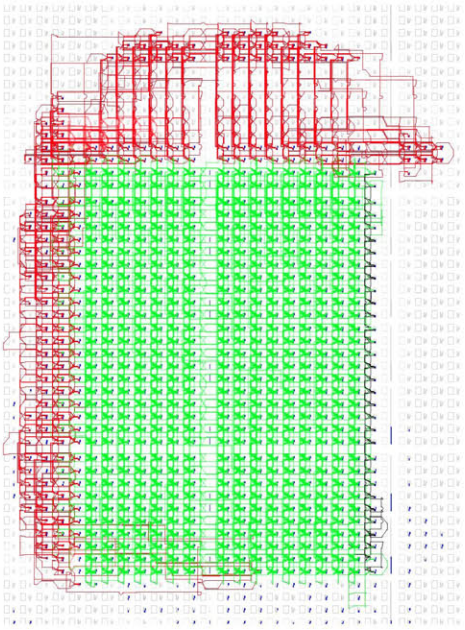

Fig. 9. A 16×16 PE systolic array as seen in FPGA Editor, with nets for input selectors in red, output selector in black, and PEs in green.

TABLE II
AVERAGE SAE AND EVOLUTION TIME (LOWER IS BETTER)

		Noise level			Time
		5%	10%	20%	(s)
(1+8)-EA	65 536 evals	29 253	56 751	114 794	1.04
	131 072 evals	20 697	37 910	95 783	2.07
	262 144 evals	14 770	28 164	79 481	4.10
8×(1+1)-EA	65 536 evals	10 749	24 131	67 087	0.80
	131 072 evals	7 824	18 465	51 050	1.61
	262 144 evals	6 317	14 801	41 361	3.23

obtained after evaluating 262 144 candidate solutions is, in average, half the one obtained with the former EA.

Additionally, by restricting mutations to a single column, the overall evolution time has been reduced by more than 20%.

Fig. 11 represents the median and 25%–75% quartiles for both former and current EAs for a 5% noise level, extended to twice the evolution time (524 288 evaluations), showing that the former one quickly gets stuck at sub-optimal solutions so continuing the evolution will barely improve the results, whereas the current one is able to reach the same results about 4 times faster and does not get stuck so prematurely.

C. Time breakdown

Table III shows the contributions of each of the parts of the evolution described in section I to the total evolution time, both for 8 candidates (which are evaluated in parallel) and the corresponding to a single candidate.

As can be seen, the time spent dynamically reconfiguring the 8 filters with the new configuration is similar to the time spent evaluating them. The time overhead due to execution of the EA in software (mutation and selection) is small, about

Fig. 10. Top: 128×128 px *Lena* image, used as training reference. Middle: the same image with 5% (left), 10% (center), and 20% (right) salt and pepper noise, used as training input. Bottom: result of filtering the middle row images with evolved filters. Array size is 8×8.

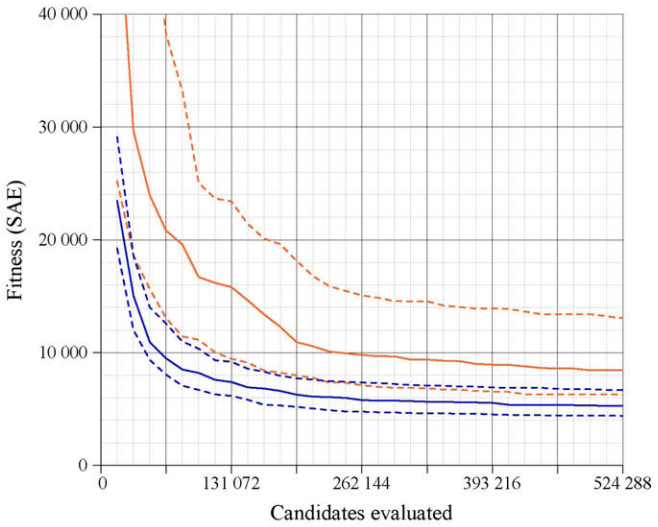

Fig. 11. Median (solid) and 25%–75% quartiles (dashed) comparing the former EA (orange) with the current one (blue), both for 5% noise level.

15% of the total.

Implementing a larger number of filters will not reduce the evolution time proportionally, since only the evaluation is performed in parallel, whereas reconfiguration, mutation, and selection are sequential.

V. CONCLUSIONS

Systolic array topologies provide a compact and resource-efficient solution for EH implementations: if a CGP architec-

TABLE III
EVOLUTION TIME BREAKDOWN (MICROSECONDS)

	×8	×1	%
Mutation	12.8	1.60	13%
Reconfiguration	41.2	5.16	42%
Evaluation	42.2	5.27	43%
Selection	1.4	0.18	2%
Total	98.5	12.31	

(81 200 evaluations per second)

ture had been used in this work, the size of a single PE would have grown from 4 slices to around 20 due to the resource overhead introduced by the multiplexers at the input of each PE.

The adoption of the LUT-based reconfiguration methodology described in [2] has further compacted the array size, from the 5 CLBs per PE in [6] to only 2. Furthermore, this methodology has reduced the reconfiguration time about 20 times, providing much faster evolutions.

The compacity of systolic arrays allows great processing speed, which has been tested to up to 500 MHz, as well as permitting the implementation of more processing arrays in the design, from one 8×7 array reported in [6] or three 4×4 arrays in [18], up to eight 8×8 arrays in a more reduced area. This parallelization, the increase of frequency of operation in the processing arrays, and the improved reconfiguration methodology allow testing over 80 000 filters per second (including reconfiguration and evaluation times for each solution).

Additionally, the use of a multithreaded EA has greatly improved the resulting quality of the obtained filters by reducing the chances of getting stuck at local minimums, allowing to make shorter evolutions for the same result. Furthermore, this EA presents advantages in scalability, since it is easily distributable among multiple separate processing nodes, as proposed in [19].

The optimization of the processing architecture speed, increase in maximum array size, speed-up of the reconfiguration methodology, parallelization of systolic arrays, and improvement of the EA, all combined, have allowed shortening the evolution time from 128 seconds reported in [5] to less than 1 second (for 65 536 candidate evaluations) with better results in the obtained filter, and showing excellent results after 3.23 seconds (for 262 144 evaluations). Therefore, the usability of such techniques in algorithms which require faster dynamic adaptation to varying situations (including fault recovery, noise level or other external conditions) has been significantly improved.

The low FPGA resource usage may allow implementing this system in smaller FPGA models, or adding even more systolic arrays to the system (although that would not improve the reconfiguration time, only the filtering time). This methodology simplifies this process, also allowing to make the arrays larger or smaller in an easy way.

Also, given that the processing element library does not need to be pre-synthesized anymore, it would be easy to extend it with new functions, such as step functions or conditional operations.

ACKNOWLEDGMENTS

This work was partially supported by the Spanish Ministry of Economy and Competitiveness under the project REBECCA, with reference TEC2014-58036-C4-2-R, and the FPI grant program of the aforementioned Ministry.

REFERENCES

[1] J. F. Miller, "An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach," in *Proceedings of the Genetic and Evolutionary Computation Conference*, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds., vol. 2. Orlando, Florida, USA: Morgan Kaufmann, 13-17 Jul. 1999, pp. 1135–1142.

[2] R. Dobai, K. Glette, J. Torresen, and L. Sekanina, "Evolutionary digital circuit design with fast candidate solution establishment in field programmable gate arrays," in *Evolvable Systems (ICES), 2014 IEEE International Conference on*, Dec 2014, pp. 85–92.

[3] *7 Series FPGAs Configurable Logic Block (UG474)*, Xilinx, Inc., 2014.

[4] H. Kung and C. Leiserson, *Systolic Arrays for (VLSI)*, ser. CMU-CS. Carnegie-Mellon University, Department of Computer Science, 1978.

[5] R. Salvador, A. Otero, J. Mora, E. de la Torre, T. Riesgo, and L. Sekanina, "Evolvable 2D computing matrix model for intrinsic evolution in commercial FPGAs with native reconfiguration support," in *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on*, June 2011, pp. 184–191.

[6] A. Gallego, J. Mora, A. Otero, E. de la Torre, and T. Riesgo, "A scalable evolvable hardware processing array," in *Reconfigurable Computing and FPGAs (ReConFig), 2013 International Conference on*, Dec 2013, pp. 1–7.

[7] Z. Vašíček and L. Sekanina, "An evolvable hardware system in Xilinx Virtex II Pro FPGA," *International Journal of Innovative Computing and Applications*, vol. 1, no. 1, pp. 63–73, 2007.

[8] A. Otero, R. Salvador, J. Mora, E. de la Torre, T. Riesgo, and L. Sekanina, "A fast reconfigurable 2D HW core architecture on FPGAs for evolvable self-adaptive systems," in *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on*, June 2011, pp. 336–343.

[9] *Partial Reconfiguration User Guide (UG702)*, Xilinx, Inc., 2013.

[10] A. Otero, E. de la Torre, and T. Riesgo, "Dreams: A tool for the design of dynamically reconfigurable embedded and modular systems," in *Reconfigurable Computing and FPGAs (ReConFig), 2012 International Conference on*, Dec 2012, pp. 1–8.

[11] C. Beckhoff, D. Koch, and J. Torresen, "Go Ahead: A partial reconfiguration framework," in *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*, April 2012, pp. 37–44.

[12] K. Glette, J. Torresen, and M. Hovin, "Intermediate level FPGA reconfiguration for an online EHW pattern recognition system," in *Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference on*, July 2009, pp. 19–26.

[13] *Virtex-5 FPGA User Guide (UG190)*, Xilinx, Inc., 2012.

[14] *Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet (DS083)*, Xilinx, Inc., 2011.

[15] *Virtex-5 FPGA Configuration User Guide (UG191)*, Xilinx, Inc., 2012.

[16] *Partial Reconfiguration of a Hardware Accelerator on Zynq-7000 All Programmable SoC Devices (XAPP1159)*, Xilinx, Inc., 2013.

[17] *LogiCORE IP XPS HWICAP (UG586)*, Xilinx, Inc., 2010.

[18] A. Gallego, J. Mora, A. Otero, R. Salvador, E. de la Torre, and T. Riesgo, "A novel FPGA-based evolvable hardware system based on multiple processing arrays," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, May 2013, pp. 182–191.

[19] J. Vazquez, B. Lopez, J. Valverde, E. de la Torre, and T. Riesgo, "Collaborative evolution strategies on evolvable hardware networked elements," in *Design of Circuits and Integrated Systems (DCIS), 2014 Conference on*, Nov 2014, pp. 1–5.