A Scalable Evolvable Hardware Processing Array

Ángel Gallego, Javier Mora, Andrés Otero, Eduardo de la Torre, Teresa Riesgo
Center of Industrial Electronics - CEI
Universidad Politécnica de Madrid
Madrid, Spain
e-mail: {angel.gallegog, javier.morad, joseandres.otero, eduardo.delatorre, teresa.riesgo}@upm.es

Abstract— **Evolvable hardware (EH) is an interesting alternative to conventional digital circuit design, since autonomous generation of solutions for a given task permits self-adaptivity of the system to changing environments, and they present inherent fault tolerance when evolution is intrinsically performed. Systems based on FPGAs that use Dynamic and Partial Reconfiguration (DPR) for evolving the circuit are an example. Also, thanks to DPR, these systems can be provided with scalability, a feature that allows a system to change the number of allocated resources at run-time in order to vary some feature, such as performance. The combination of both aspects leads to scalable evolvable hardware (SEH), which changes in size as an extra degree of freedom when trying to achieve the optimal solution by means of evolution. The main contributions of this paper are an architecture of a scalable and evolvable hardware processing array system, some preliminary evolution strategies which take scalability into consideration, and to show in the experimental results the benefits of combined evolution and scalability. A digital image filtering application is used as use case.**

Keywords—evolvable hardware, scalability, dynamic and partial reconfiguration, FPGAs

I. Introduction

Evolutionary algorithms are suitable methods to get solutions to problems where the quality of the solution can be measured by a function, typically called fitness function, which is to be maximized or minimized. Genetic algorithms [1][2] are types of evolutionary algorithm which, as in biology, propose iterative candidate solutions whose fitness are evaluated and, after a selection of the best candidate(s), new offspring is generated based upon modifications of the previous solutions. Proper selection of fitness functions, candidate selection and adequate mutation or crossover rules for generating new candidates may produce results that are far better than other space exploration techniques.

If circuit design is the goal of an evolutionary algorithm and the fitness function is somewhat related to the functionality to be achieved by the circuit, then the system is referred as evolvable hardware [3][4]. It may be used off-line, as an alternative to traditional design tools, although for autonomous systems, it is far more attractive the on-line version, where the design is, at its utmost level of autonomy, providing additional capabilities such as self-adaptiveness or self-healing. One requirement for these interesting properties is intrinsic evolution [5], where the device that holds the circuits to evaluate during evolution is the same device that will be used for normal operation.

FPGAs and their reconfiguration capability are excellent candidates for EH systems. The integration levels they achieve permit to integrate both the reconfigurable device on which to evolve, and the EA. Some approaches, like *Cartesian Genetic Programming* (CGP) [6] use combinations of logic and interconnects that are geared towards generic design at gate level, whereas in our approach we use a reconfigurable processing array composed of tiny processing elements (PEs) which, all together, perform some computation. The selection of the library of PEs determines which type of applications is targeted.

Most evolutionary approaches, as well as most EH systems, with exceptions like [7], are based on fixed-size systems. One of the limitations for this is that, this way, the genotype (the coding of the properties of the system) yields to fixed-size implementations (also called as phenotypes). In our case, we are proposing a scalable reconfigurable processing array, which may grow or shrink in dimensions in order to adapt to different problems and changing conditions. Some authors combine the EH with the concept of 'development', as presented in [8], in the sense that the adaptation process of an individual can increase skills only achievable by development of the individual's tissues, growing in size.

However, from a practical point of view, the combination of both techniques is not straightforward. First, the EA must accommodate individuals with variable-length genotypes. The problem is not just how to allocate more room for storing longer genomes, but, for instance, to accommodate different mutation rates to different sizes. On the other side, the scalability on the underlying circuit to be built (the phenotype) must be based on a well-defined and consistent architecture which is able to scale up with no problems. Furthermore, the system should be based on DPR, as the one presented in [9] or the system used in this work, above other techniques such as the *Virtual Reconfigurable Circuits* (VRC) [10], which present a high area overhead and are not suitable for growing up in the number of elements.

The aim of this paper is to show the architecture designed for scalability, which is the main contribution to the state-of-the-art evolvable hardware systems, laid within the complete SEH architecture, which has been entirely mapped on an FPGA. EA adaptations as well as architectural transformations to enable scalability are shown. Some details on the tool that helps on building up such system are also presented. Finally, experimental results will show the added benefits of combining such techniques.

II. Architecture of the Scalable Processing Array

The system described in this work corresponds to the evolution of a previous one that was presented in [11], which is briefly described in the next sub-section. The final architecture that results when scalability is introduced is described in detail afterwards.

A. *Initial evolvable non-scalable system*

The initial system is composed of three main elements: an embedded microprocessor (a MicroBlaze) that is in charge of running the evolutionary algorithm (EA) and selecting a proper configuration for the filter; the reconfiguration engine, an enhanced version of Xilinx's HWICAP, able to configure the same bitstream in different positions, and in charge of changing by DPR the candidate circuit according to the configuration obtained by the EA; and the evolvable hardware circuit, that consist of a systolic array of processing elements (PEs), each of one performs a simple operation per clock cycle (such as addition, subtraction, logic shift of bits, maximum, minimum, average, and others). The processing array is the dynamic or reconfigurable part of a peripheral, which includes in its static part all the logic for the control of the process, several FIFOs that prepare the inputs of the array, and an evaluation unit in charge of calculating the fitness value, which measures the quality of the solution and is used in the evolutionary algorithm when evolving the circuit. In this case, the fitness function is the summation, pixel by pixel, of the absolute errors (SAE) between the filtered output image and a reference noise-free image, so the lower the fitness, the better the configuration for that task. Finally, the system includes a Compact Flash memory that stores the different configurations of the reconfigurable elements and the training images. All this architecture is shown in Fig. 1.

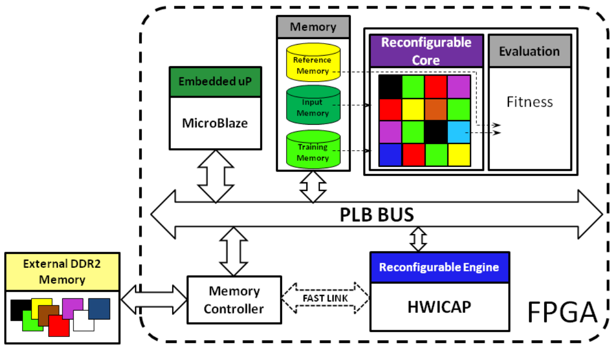


Fig. 1 Architecture of the evolvable hardware system.

The processing array is more deeply detailed in Fig. 2. As can be observed, each PE has two inputs (north and west) and two outputs (south and east), that allow them to be connected in a tiled form. In order to feed the processing array, there are several input multiplexers that select one out of nine possible input pixels stored in a 3x3 window which slides over the whole image going pixel by pixel each clock cycle, and corresponds to the circuit shown in Fig. 3. The selection of the input multiplexers is changed by the EA, and they are part of the configuration of the candidate circuit. The selection of the output is done by a multiplexer placed at the east side of the array. This selection is also determined by the EA.

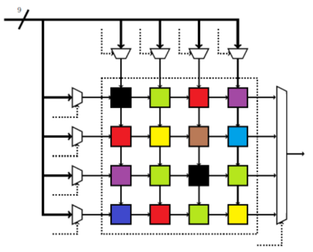


Fig. 2 Structure of the non-scalable processing array, with the input multiplexers, the output selector, and the processing elements.

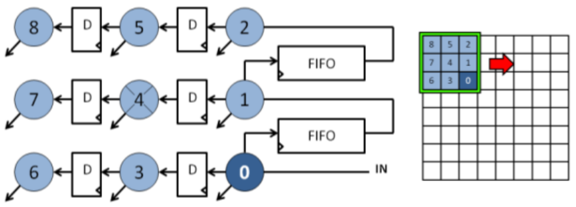


Fig. 3 Circuit implemented to create the sliding window, with the 9 available input pixels, and the representation of the window sliding over the image.

Regarding the evolutionary algorithm, an algorithm based on *Cartesian Genetic Programming* (CGP) is used, following a *(1+λ) Evolution Strategy* with 1 parent and λ (set to 8 in this case) offspring per generation. The offspring are generated by mutating the parent, which is the best candidate of the previous generation (the circuit with lower fitness). More details can be found in [12].

B. *Scalable system*

In order for the array to be scalable, several modifications had to be done in the architecture, all of them related with the reconfigurable peripheral and the processing array in particular, while keeping the rest of the system as before. On one hand, the structure of the PEs makes them suitable to be scaled, as they can be connected one after the other in a tiled way without any difficulty. But, on the other hand, the input and output data multiplexers, which were easily reconfigured in the previous design just by changing the value of the control signal stored in an internal register, now become an obstacle. Regarding the input multiplexers, if the size is increased in any direction, the new row or column will need new input multiplexers. This is not a problem for the data input signals, but it is a problem for the selection signals, since an extra set of selection signals is required for every new row or column. The output multiplexer has a similar but not equal problem, since it is only affected when rows are increased, and it implies adding a new input for this multiplexer and, if needed, an extra control bit.

The lack of scalability was solved by making these elements reconfigurable. Every multiplexer is replaced by a HW block, compatible in size with the PEs, which hardwires the selected signal into the input ports of the associated PE. This implies to design nine new reconfigurable modules, one for every selected input (these are the nine pixels of the sliding window). Candidate generation becomes slower since the former multiplexer configuration by just writing in a register, and is replaced by a module reconfiguration However, it makes the candidate generation process more homogeneous, since every gen in the genotype corresponds to a module being configured in the corresponding positions in the phenotype.

With this modification of the blocks, the structure of the scalable processing array is the shown one in Fig. 4.

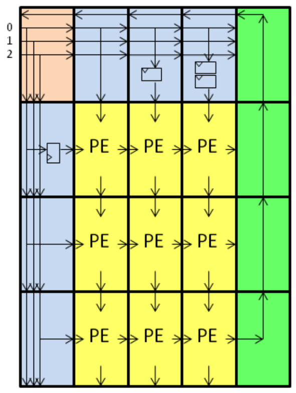


Fig. 4 Structure of the scalable processing array, with the reconfigurable input multiplexers (blue), the output selector (green), the array connector element (orange), and the PEs (yellow).

In order to reduce the routing logic inside each hardwired multiplexer, the circuit that implements the sliding window has been modified (Fig. 5) and some elements are now inside the hardwired blocks, so just 3 pixels are fed to the system.

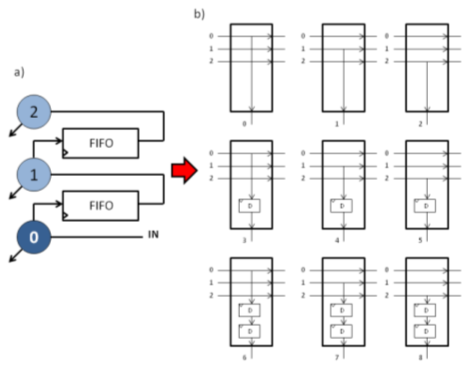


Fig. 5 New circuit to implement the sliding window (a), and the resulting hardwired multiplexers that allow the system to be fed by one of the 9 traditional window pixels (b).

The output selector consists of three blocks, which take the output from the desired element and send it back to the static part of the peripheral. To be sent back without extra elements, the horizontal multiplexers at the top have a return path for the output pixel. To join the array with the static part, an extra connector element needs to be reconfigured (just once, the first time), and it sends the three generated pixels from the window to the vertical and horizontal multiplexers, and collects the output pixel.

III. Design Considerations for Scalability

As it can be seen, the architecture for scalable evolvable hardware requires designing a relatively large amount of tiny re-allocable modules that need to maintain compatibility in their connections, allowing a tiled arrangement, basic to achieve scalability. Modules can be reallocated in different positions, and this mandates to have a reconfiguration controller able to reallocate bitstreams, by manipulating the frame addresses of the bitstream.

Another challenge in the design of these modules, especially in the case of very small ones (as in this case, where PEs have only 5 CLBs), is that it is not possible to use bus macros, and specific routing skills are required in order to achieve size-efficient routing constrained designs, with placement restrictions in all I/Os. At this point is where our DREAMS tool, introduced in [13], is a great productivity enhancement. It allows defining 'virtual borders' which are definitions of boundaries which keep compatibility between different elements, either the static part or the reconfigurable modules which are to be placed into compatible positions. Starting from a GUI which produces an XML description of the system: areas, modules, connections, etc., a complete script is produced in order to synthesize and place and route all required elements, ending up with a collection of bitstreams to be used either offline at design time or in an offline application. The DREAMS tool is based on RapidSmith [14], and contains a custom router than handles restrictions and ensures that routing will be compatible between any pair of adjacent compatible modules.

The architecture presented in the previous section is of utmost importance to achieve scalability. The DREAMS tool, on the other side, permits to speed up designs so that, compared with hand-made designs, may reduce design time from several days to a couple of hours. There is still a third element to be modified in order to permit scalability: the evolutionary algorithm. It must be designed such that it supports variable genomes, so all storage elements must keep track of the size of the processing array and all its corresponding genes (PEs in every array position, input multiplexers and output multiplexer). However, the main problem is not this one, but to find a coherent evolution strategy that accommodates to variable size elements. There are some critical factors, derived from the growth of the design exploration which may impact negatively on the performance of the evolvable system. Some of these considerations are shown in the next section.

IV. Run-Time Design Space Exploration

While it is clear that scalability gives an extra degree of freedom in the evolvable system, it also has to be tackled appropriately. Evolutionary algorithms are normally tuned to accommodate to different design space sizes, and the problem is that scalable hardware has a scalable design space to explore. Accordingly, some parameters like the number of generations to stop evolution, or the mutation rate may differ importantly from simple systems than for complex ones, where convergence may occur at much higher number of generations. In the experimental results section, an analysis of the number of generations and a scalable-compatible dynamic stop-criterion technique are shown. By some extra experimentation, it has been observed that variable mutation rate along evolution may improve results (either converge faster or reach better quality). This analysis is left for future research.

Apart of the need of tuning the EA, it is required to decide when and how to scale up or down. Resource occupation obviously indicates that smaller circuits are better, and they even converge faster, but resulting quality, fault tolerance and other factors are worsened. This trade-off may be solved by two generic possibilities: a) to use multi-objective evolution, where size is included in the fitness function with a coefficient that balances weights between size and functionality; or b) to use the concept of development, starting from a small size and growing up in order to accommodate to better performance.

Among the aforementioned possibilities, we have used the second option, development based, since handling different sizes from one generation into the next one, or even between candidates of the same generation, might have an important impact on reconfiguration time, which would slow down the evolution importantly.

Thus, development is used in the proposed approach. Starting from small size arrays, if the fitness value is not of sufficient quality, size is increased and evolution is restarted. Here, one possibility is to start from scratch, not reusing the previous evolved circuits of smaller size, or to adapt them as the starting point of evolution for the new size. The adapted candidate to resume evolution with bigger size is obtained by filling the new column with pass-through blocks (one of the PEs is exactly that, extending data one column) or by just adding one row at the bottom (since the output is on the east side, there is no need to extend to the south). In the experimental results, a comparison of both types of development will be done, showing that one technique or the other one are more effective depending on the array size.

V. Experimental Results

The validation of the proposed architecture has been carried out considering the three parameters mentioned above: the area or resource utilized by the system depending on the size; the performance of the different possible sizes; and the time that the circuit needs to self-adapt at run-time. Those results lead to different evolutionary strategies that can be used depending on the complexity and the requirements of the tasks. The experimental results provided in this paper were obtained with the system implemented in a medium size Xilinx Virtex-5 LX-110T FPGA, with the MicroBlaze, the reconfiguration engine, and the systolic array working at 100 MHz.

In all the experiments carried out in this paper, an input image with 5% salt and pepper noise has been used. With this image the median filter obtains a SAE fitness of 89345, which is higher than the fitness obtained with our system (for any size with the exception of a 1x1 filter). This comparison was made in [12], and since the subject of the paper is the scalability of the system, the results are no longer compared with the median filter output, but they are compared with the results with different sizes.

A. Area and resources utilization

In Fig. 6, a snapshot of the floor-planning of the system can be seen, with an empty reconfigurable area where the systolic array can grow up and down. The elements of the array have different sizes, depending on the type. Thus:

- The array connector element, which feeds the array with the input pixels and collects the output pixel, employs two CLB columns wide by half clock region height (10 CLBs), with a total utilization of 20 CLBs.
- The horizontal muxes, which feed the first row of PEs, occupy one CLB column wide by half clock region height each one (10 CLBs).
- The vertical muxes, in charge of feeding the first column of PEs, use two columns of CLBs in width by a fourth of the clock region in height each one (10 CLBs).
- Each of the PEs and any of the elements that compose the output selector occupy 5 CLBs, one CLB column by a fourth of the clock region.

Regarding these values, the next formula specifies the total number of CLBs occupied by an N height by M width processing array.

$$Total\ CLBs = 20 + (N + M) \times 10 + N \times (M + 1) \times 5$$

The reconfigurable area has been limited to a maximum array size of 7x7 PEs, which means a total number of 440 CLBs reserved for the systolic array. The static part of the peripheral utilizes 1231 slices, 2766 FFs and 3158 LUTs, while the whole system (including the static part) employs 5116 slices, 11475 FFs and 10691 LUTs, corresponding to the 29%, 16% and 15% of the total available ones respectively.

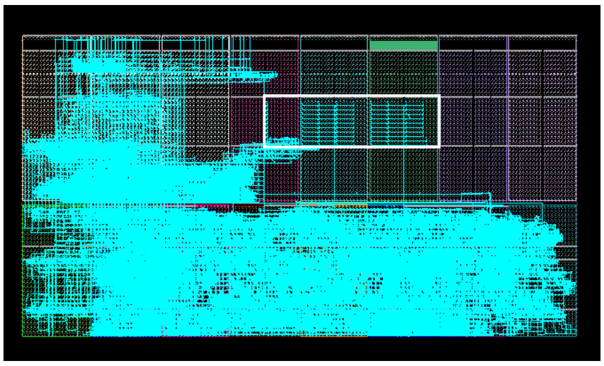


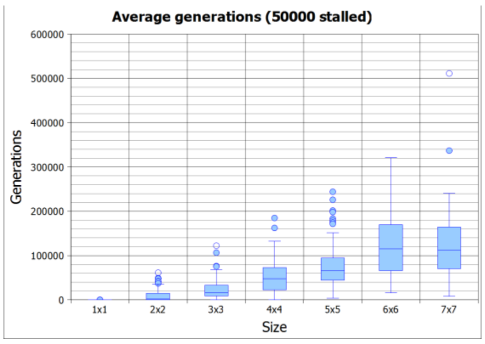
Fig. 6 Layout of the implemented system, with the reconfigurable area highlighted in white.

B. Evolution time for fixed size arrays

For fixed size systems as the initial one, a predefined fixed number of generations is typically used. But when dealing with different sizes, the higher the height or the width of the array, the larger the chromosome, and that means the search space becomes larger. This makes the evolution stage longer, and determining the optimal number of generations for each size is not trivial. In order to determine an appropriate duration of the evolutions, it was decided to stop evolution after 25000 or 50000 generations with no fitness improvement. The average of the results obtained from 100 independent evolution runs for all the possible square array was obtained, corresponding to Fig. 7.

As it can be seen, when the number of processing elements is low, the search space is quite small, and a solution can be obtained in a few generations. But with the biggest sizes (6x6 and 7x7), the number of generations increases significantly, which means it is harder for the system to find a proper configuration of PEs.

(a)



(b)

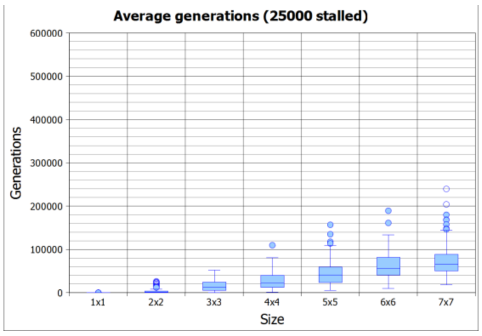


Fig. 7 Average number of generations needed to evolve until the result does not change in 50000 consecutive generations (a) and 25000 consecutive generations (b).

In Fig. 8 the statistical fitness distribution of a 7x7 array evolution are represented at the generation 75000 (average generation obtained with the 25000 non-varying experiment) and at the generation 150000 (average generation obtained with the 50000 non-varying experiment). Since stopping the evolution at 150000 would last twice compared with evolving until 75000 generations, and also considering that the improvement provided in those extra generations is not significant, the average generation obtained with the experiment with 25000 non-changing generations has been selected to be the duration of each evolution. In the case of a rectangular array, the duration is the one as the square array that can contain it, i.e. the square array size $max(height, width)$ x $max(height,\ width)$, oversizing the needed amount of evolution time in some cases.

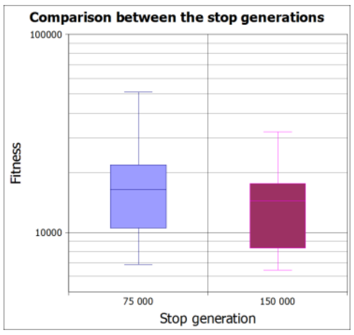


Fig. 8 Fitness comparison between the stop generations obtained with the experiments with 25000 non-changing generations (75000 generations) and with the 50000 one (150000 generations).

C. *Performance analysis for different sizes*

Fig. 9 shows the 3D representation of the average fitness value of 50 independent evolutions, starting from a random chromosome in each evolution, and evolving the system for the aforementioned size-dependent number of generations. As can be seen, the scaling of the array in height, width, or both, makes the performance of the system increase.

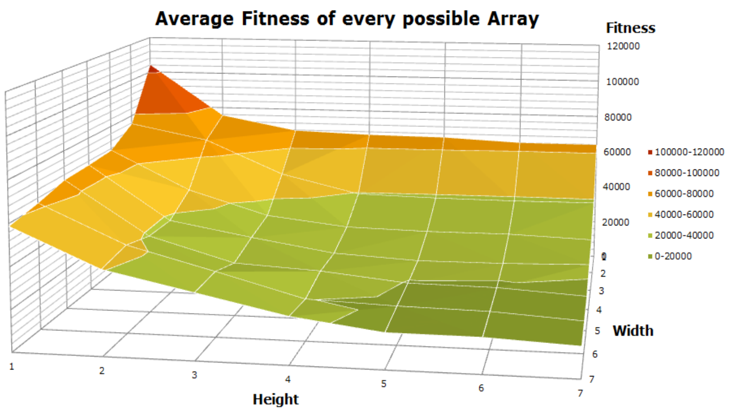


Fig. 9 Tridimensional surface formed by the average fitness of 50 independent evolutions with every possible array size.

It can be observed that arrays *1*x*N* or *N*x*1* do not produce a big improvement when scaling up, apart from the case of 1x1, due to the lack of possibilities of interaction between the different input pixels of the array. In the case of either a 2x3 or a 3x2 array, the fitness obtained is better than the 1x6 or 6x1 arrays, which use the same number of processing elements, but as they are disposed in a line, there is only one propagation path, limiting thus the complexity in the processing. The resulting images of the best case for every square array and the original noise-free image are shown in Fig. 10.

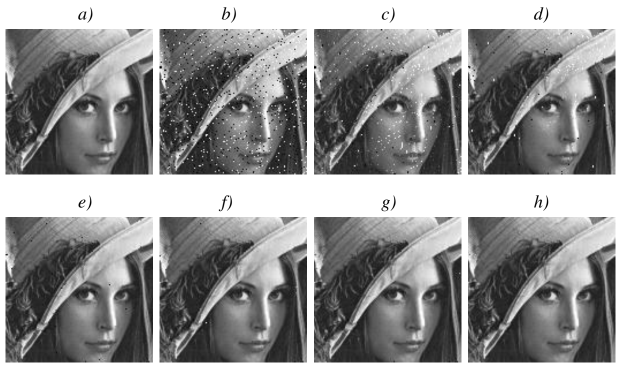


Fig. 10 Output image of the best filter configuration for every square array: the original image (a), and from 1x1 to 7x7 (from (b) to (h) respectively). The 1x1 result also corresponds to the noisy input image, as the best configuration obtained in that case is a copy filter.

D. *Evolutionary strategies for scalability*

Another thing to consider is the development strategies that can be applied in this system. In this work the development is carried out by evolving the system sequentially, from lower sizes to larger ones, in order to guide the evolutionary algorithm in the bigger arrays, where the search space is bigger. In Fig. 11 the average fitness of 50 independent evolutions using this sequential evolution and with the classical evolution starting from a random chromosome are shown.

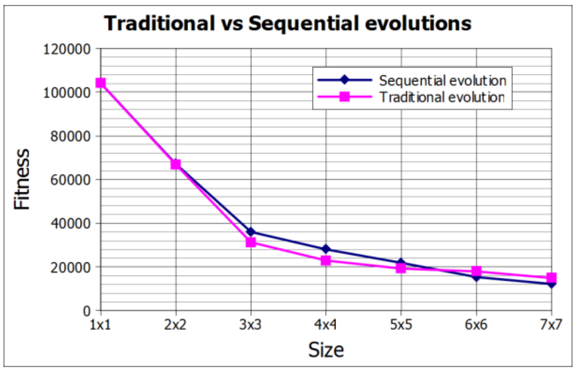


Fig. 11 Traditional vs. incremental evolution. Average fitness values for 50 independent runs.

It can be seen how the smaller arrays obtain better results starting from scratch rather than evolving sequentially. Considering also the accumulated times spent in the case of the sequential evolution, it seems better to evolve sizes up to 5x5 from zero. But, for 6x6 and 7x7, the results obtained are better than in the traditional approach. That means that the guidance provided by starting from a working configuration works with larger search spaces.

A proper application of this methodology in order to save time is combine both approaches, i.e. evolve from a random chromosome a 5x5 array, and from that array obtain the 6x6 and 7x7 arrays with the sequential evolution applying the development strategy. In Fig. 12 the results of going from a 5x5 to a 7x7 with this combined methodology are compared with the previous presented ones, and it can be seen how it behaves better than in the other cases, due to the fact that the sequential evolution starts from a configuration with better performance, and also the evolution lasts less number of generations than with the sequential evolution from a 1x1 array.

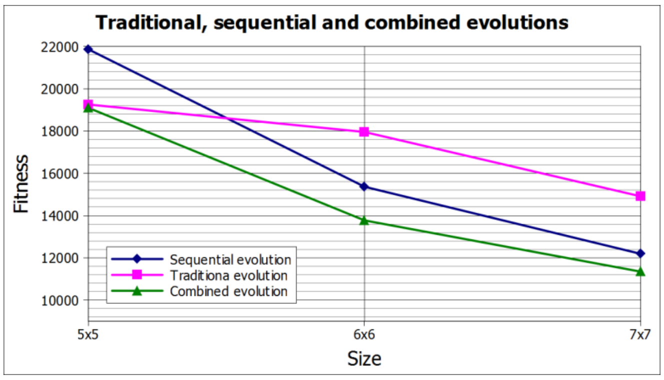


Fig. 12 Average fitness obtained in 50 independent evolutions with the traditional evolutionary strategy, the sequential strategy, and the combination of both (evolving the 5x5 array from a random chromosome, and the 6x6 and 7x7 applying the development strategy).

E. Comparison between multiple arrays versus scalable array

As was presented in [15], the total performance of the filter can be enhanced by having multiple processing arrays arranged in series or cascaded. In that case, the output image of one array is the input image of the next one, and based on the adaptability of the system, if the cascaded filters are evolved in that configuration, each one adapts to the behavior of the previous one, getting a higher quality output image. So regarding just the performance, Fig. 13 shows the fitness obtained in the case of different sizes of the array (scalable array) following the evolutionary strategy based on development explained before.

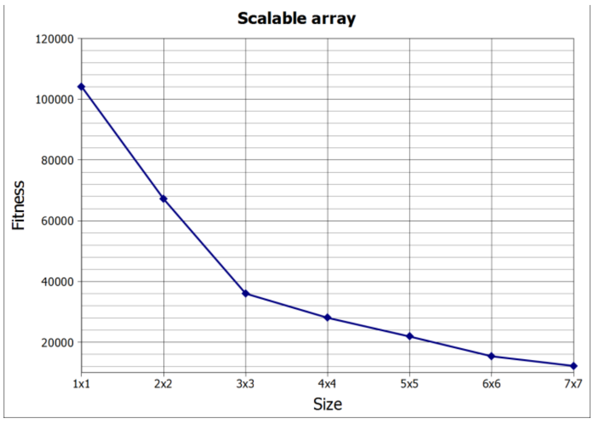


Fig. 13 Average fitness of 50 independent evolutions for every square array.

In Fig. 14, it can be seen more in detail how, for the 4x4 array, the results are more or less the same value as at the first stage of the multiple arrays, since the processing arrays implemented in that work were size 4x4. As shown, similar fitness value is obtained with two stages of the cascaded filter and with a 5x5 PEs array, and in the rest of the cases the fitness is always much lower in the case of the scalable array. The result of three 4x4 cascaded arrays is clearly improved by the scalable 7x7 array, with similar number of PEs (48 vs. 49 PEs)

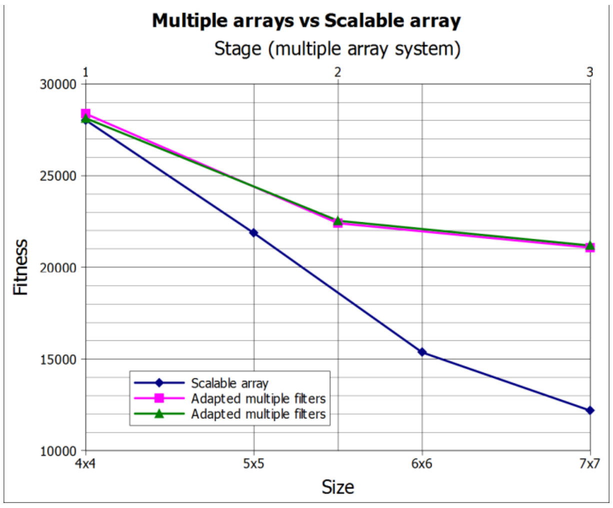


Fig. 14 Comparison between multiple arrays (stages 1, 2 and 3) evolved with two adaptive techniques, and the fitnesses obtained with scalable arrays, form size 4x4 to 7x7.

However, the total amount of generations needed to obtain two 4x4 arrays is $2 \times 31000 = 62000$ generations, which is lower than the number of generations that are needed in the case of the scalable array with the sequential evolution, which is 102000, due to the accumulation by evolving smaller size arrays.

If we compare real resource occupation rather than number of PEs, the comparison is more in favor of the scalable solution. For instance, the number of CLBs used by a 5x5 array is 270, while the number of CLBs used in the processing arrays (if they were implemented with this new architecture), would be $2 \times 200 = 400$, which is a 48% more of CLBs used, and that is without considering the extra logic needed in the static part of the controller for each array. But on the other hand, having several arrays can be used to perform several different tasks at one time, as for instance noise removal and edge detection.

VI. Conclusions and Future Work

Along this paper, a scalable evolvable hardware processing array has been introduced, giving some details and hints about the usability and the possibilities of the system. It has been shown how bigger arrays lead to qualities never obtained until now, but with a penalization in the time needed to adapt the system at run-time, which is bigger with the largest arrays. Also an evolutionary strategy has been proposed, based on the concept of development, which obtains better results by evolving sequentially the system from one size to a bigger one. But there are more parameters to analyze, such as the complexity of the task, for instance with more noisy input images, and the fault tolerance of the system, that was explored in [16] for the case of the non-scalable architecture, and now it is expected to be enhanced due to the extra degree of freedom introduced. Another trend to work in is the integration of the scalable array with the multiple processing arrays system presented in [15], obtaining a fully scalable architecture, and also the development of the proper evolutionary algorithm in charge of deciding when it is needed to scale, and how to do it, whether increasing the number of processing arrays or the size of one of them.

Acknowledgment

This work was supported by the Spanish Ministry of Economy and Competitiveness under the project DREAMS (Dynamically Reconfigurable Embedded Platforms for Networked Context-Aware Multimedia Systems) with number TEC2011-28666-C04-02.

References

[1] de Garis, H.; "Evolvable hardware genetic programming of a Darwin machine," *Artificial neural nets and genetic algorithms*. Springer Vienna, 1993.

[2] Oreifej, R.; Sharma, C.; DeMara, R.F.; “Expediting GA-Based Evolution Using Group Testing Techniques for Reconfigurable Hardware,” Int. Conf. on Reconfigurable Computing and FPGAs (ReConFig 2006), IEEE, 2006, pp. 1-8.

[3] Sakanashi, H.; Iwata, M.; Keymulen, D.; Murakawa, M.; Kajitani, I.; Tanaka, M.; Higuchi, T.; " IEEE International Conference on Evolvable hardware chips and their applications," 1999, vol.5, no., pp.559-564.

[4] Zdenek, V.; Sekanina, L.; "An evolvable hardware system in Xilinx Virtex II Pro FPGA," *International Journal of Innovative Computing and Applications* 1.1 (2007): 63-73.

[5] A.M. Tyrrell, G. Hollingworth, S.L. Smith, “Evolutionary strategies and intrinsic fault tolerance” Proc. 3rd NASA/DoD Workshop on Evolvable Hardware. EH-2001, IEEE Comput. Soc, pp. 98-106.

[6] Miller, Julian F., and Peter Thomson. "Cartesian genetic programming." *Genetic Programming*. Springer Berlin Heidelberg, 2000. 121-132.

[7] Torresen, J., "Scalable evolvable hardware applied to road image recognition," *Evolvable Hardware, 2000. Proceedings. The Second NASA/DoD Workshop on* , vol., no., pp.245,252, 2000.

[8] Torresen, J.; "A scalable approach to evolvable hardware," Genetic programming and evolvable machines 3.3 (2002): 259-282.

[9] Torresen, J.; Senland, G.A.; Glette, K., "Partial Reconfiguration Applied in an On-line Evolvable Pattern Recognition System," NORCHIP, 2008. , vol., no., pp.61-64, 16-17 Nov. 2008.

[10] L. Sekanina, “Virtual Reconfigurable Circuits For Real-World Applications Of Evolvable Hardware” Proc. of the 5th international Conf. on Evolvable systems: from biology to hardware. ICES 2003, vol. 2606, pp. 186-197.

[11] Otero, A.; Salvador, R.; Mora, J.; de la Torre, E.; Riesgo, T.; Sekanina, L.; "A fast Reconfigurable 2D HW core architecture on FPGAs for evolvable Self-Adaptive Systems," NASA/ESA Conference on Adaptive Hardware and Systems (AHS), 2011.

[12] Salvador, R.; Otero, A.; Mora, J.; de la Torre, E.; Riesgo, T.; Sekanina, L.; "Self-Reconfigurable Evolvable Hardware System for Adaptive Image Processing," Computers, IEEE Transactions on , vol.62, no.8, pp.1481,1493, Aug. 2013.

[13] Otero, A.; de la Torre, E.; Riesgo, T., "Dreams: A tool for the design of dynamically reconfigurable embedded and modular systems," *Reconfigurable Computing and FPGAs (ReConFig), 2012 International Conference on* , vol., no., pp.1,8, 5-7 Dec. 2012.

[14] Lavin, C.; Padilla, M.; Lamprecht, J.; Lundrigan, P.; Nelson, B.; Hutchings, B., "RapidSmith: Do-It-Yourself CAD Tools for Xilinx FPGAs," *Field Programmable Logic and Applications (FPL), 2011 International Conference on* , vol., no., pp.349,355, 5-7 Sept. 2011.

[15] Gallego, Á.; Mora, J.; Otero, A.; de la Torre, E.; Riesgo, T.; Salvador, R; "A Novel FPGA-based Evolvable Hardware System based on Multiple Processing Arrays," Reconfigurable Architectures Workshop (RAW), International Parallel & Distributed Processing Symposium, IPDPS, 2013.

[16] Salvador, R.; Otero, A.; Mora, J.; de la Torre, E.; Sekanina, L.; Riesgo, T.; "Fault Tolerance Analysis and Self-Healing Strategy of Autonomous, Evolvable Hardware Systems," International Conference on Reconfigurable Computing and FPGAs (ReConFig), 2011.