On Robust Evolution of Digital Hardware

Tobias Knieper, Bertrand Defo, Paul Kaufmann, and Marco Platzner

Abstract In this paper we investigate whether multi-objective evolution of digital hardware components has advantages over single-objective evolution in terms of convergence and robustness. To that end, we experimentally compare a standard genetic algorithm to several multi-objective optimizers on a set of test problems. The results show that, for more complex test problems, the multi-objective optimizers TSPEA2 and NSGAII indeed outperform the single-objective genetic algorithm as they more often evolve correct circuits, and mostly with less computational effort.

1 Introduction

Self-adaptive and self-optimizing systems are able to react to changes in the environment and the internal system state autonomously. Systems with such self-X properties find applications in, for example, highly complex scenarios where classical methods fail, or in scenarios which require autonomous operation for long mission periods. To design such systems, often principles from biology or sociology are transferred into engineering domains and combined with modern hardware and software technology to form what is denoted as *organic computing*.

In our work, we focus on organic computing methods to develop hardware components. More than a decade ago, the emergence of reconfigurable hardware architectures together with natural computing methods gave rise to the field of *biologically-inspired hardware,* which includes several areas [1]: *Evolvable hardware* denotes the combination of evolutionary algorithms with reconfigurable hardware to construct self-adaptive and self-optimizing hardware systems. *Embryonics* tries to apply developmental processes as found in multicellular organisms to design fault-tolerant circuits with self-repair and self-healing capabilities. *Immunotronics* uses principles of the immune system to support fault tolerance and protection for

University of Paderborn
e-mail: {tknieper, bertrand, paul.kaufmann, platzner}@upb.de

hardware circuits. Finally, *neural hardware* denotes hardware implementations of models of the nervous system.

We concentrate on evolvable hardware, a term coined by de Garis [2] and Higuchi [3] in 1993. The common denominator of all evolvable hardware approaches is the application of evolutionary algorithms directly at the hardware level. Here, hardware means both digital and analog electronic circuits, and the hardware level comprises all models of hardware, from configuration bitstreams for reprogrammable devices over netlists of gates to behavioral descriptions. In a sense, evolutionary algorithms exploit a form of collaborative computing as they keep a set of individuals and try to improve them over generations by applying biologically-inspired operators such as selection, mutation, and crossover. Especially crossover allows to pass on information between individuals.

Most work in evolvable hardware has been focusing on evolving functionally good or correct components. In contrast, we consider several objectives and include also the required hardware area and the resulting circuit speed in the evolutionary process. While a few previous approaches applied two-stage fitness functions [4, 5], we employ multi-objective evolutionary algorithms (MOEAs). There are two motivations for using MOEAs in evolvable hardware design: First, MOEAs are designed to keep diversity in the population and generate Pareto sets of circuits. Autonomous systems can switch between the solutions in the Pareto set in order to react on quickly changing resource situations and performance goals [6]. Second, MOEAs can possibly be used for a faster and more robust evolution of functionally good circuits. The argument is that putting too much selection pressure on only one objective (the functional quality of the circuit), instead of keeping the population diverse with respect to other objectives (area and speed of a circuit), one might more easily get stuck in the optimization process and, hence, need a higher computational effort to evolve components with acceptable functional quality.

In this paper, we want to investigate whether multi-objective evolution of digital hardware components has advantages over single-objective evolution in terms of convergence and robustness. To that end, we experimentally compare a standard genetic algorithm (GA) to several recent multi-objective evolutionary algorithms on a set of test problems. The paper is structured as follows: In Section 2, we present the hardware representation model that is used to encode circuit individuals, and the computation of objectives. The different evolutionary algorithms including four multi-objective optimizers are discussed in Section 3. Section 4 shows the test problems, the experimental setup and the results, before Section 5 concludes the paper.

2 Hardware Representation Model and Metrics

Cartesian genetic programming (CGP) is a very popular hardware representation model introduced in [7]. CGP is a structural hardware model, where a circuit is formed by combinational logic blocks arranged in a two-dimensional array and an interconnect (wires) between the blocks. Figure 1 presents the CGP model and its

parameters. The array consists of $n_c \times n_r$ combinational blocks, n_i primary inputs, and n_o primary outputs. The primary inputs can be connected to the inputs of any logic block in the array. A logic block in column j has n_n inputs that can be connected to the columns $j-l,\ldots,j-1$ of the array and to the primary inputs, respectively. This ensures that no combinational feedback loops are generated. A combinational block implements one out of n_f different logic functions of its inputs.

An individual is defined by its chromosome (genotype). The length of the chromosome is given by $n_c \cdot n_r(n_n+1)+n_o$. Each of the logic blocks in the array is defined by n_n+1 values, one for each input and one for the logic function. Additionally, an n_o-tuple of values selects the block outputs that are connected to the primary outputs of the array.

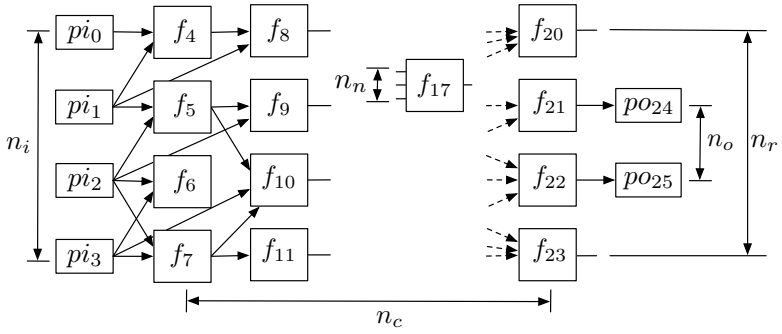

Fig. 1 The cartesian genetic programming model for hardware representation with its main parameters

The main reason for the popularity of the CGP model is its closeness to the architectures of field-programmable reconfigurable hardware arrays (e.g., FPGAs or coarse-granular arrays). The block functions can be set to simple two-input gates, to n_n-input lookup tables, or to more complex word-based arithmetic operators. The interconnect can model bit wires or busses. While the original CGP model implicitly encodes block placement, more recent CGP variants rely on only one row of blocks, i.e., $n_r = 1$ and $l = n_c$. Routing is not encoded in the CGP model, mainly to keep the genotype (chromosome length) short and, thus, to increase the efficiency of the evolutionary operators.

Generally, the genotype has to be mapped to a corresponding phenotype for evaluating the fitness. The phenotype represents the actual circuit and is achieved from the genotype by removing all blocks of the array that do not contribute to the outputs. Note that there might still be redundancy in the phenotype. An important previous result with the CGP model is that propagating redundant and currently unused structures inside the chromosomes through the search process of the evolutionary algorithm can increase the speed of convergence [7].

In this paper, we are interested in evaluating the circuits' fitness with regard to three objectives: the functional quality, the speed of the circuit, and the required

hardware area. Accordingly, we have to define three metrics to evaluate circuit fitness. Following related work in evolvable hardware, we use logic and arithmetic functions as test problems [8]. As we know the correct outputs for all input value combinations for these functions, we determine the *functional quality* as reciprocal of the summarized square error distances between the output vectors of an evolved individual c and a correct function c^*:

$$f(c) = \frac{1}{1 + \frac{1}{N}\sum_{i=1}^{N} ham(c^*(i), c(i))^2}, \tag{1}$$

where N denotes the number of test vectors and *ham* refers to the Hamming distance of two bit vectors. A correct circuit receives a functional quality of one.

We estimate the delay of a circuit by the number of wires or logic blocks on the longest path. Given the CGP model, the delay is in the range $\{0,\ldots,n_c+1\}$. A delay of zero means that the longest path of the circuit connects an input directly with an output. A delay of n_c means that the longest path traverses all logic blocks of the model, whereas a delay of n_c+1 indicates that none of the outputs is connected to an input. The fitness with respect to circuit *speed* is determined as:

$$speed(c) = 1 - \frac{delay(c)}{n_c + 1} \tag{2}$$

The speed metrics equals one for the fastest possible circuit (a circuit that maps primary inputs directly to primary outputs) and zero for a circuit that has no connection at all from primary inputs to primary outputs.

The number of logic blocks used by a circuit c, denoted as $used\_blocks(c)$, is in the range $\{0,\ldots,n_c \cdot n_r\}$. Based on this value, we define a circuit's fitness with respect to *area* as:

$$area(c) = 1 - \frac{used\_blocks(c)}{n_c \cdot n_r} \tag{3}$$

A circuit of minimal size, i.e., a circuit not using any logic block, receives an area of one, a circuit that utilizes all available logic blocks has an area of zero.

3 Multi-objective Optimizers

In this section, we review the multi-objective evolutionary optimizers SPEA2, TS-PEA2, NSGAII, and μGA that are compared in our experiments. As a reference algorithm, we use a standard single-objective genetic algorithm (*GA*). The parameters for GA are set as follows: The top 5% of the individuals are selected and transferred without any modification to the next generation. Then, we apply two-stage binary tournament as selection scheme, followed by a two-point crossover with a recombination probability of 90%, and mutation. We choose the mutation rate such that

only one combinational block or wire is mutated each time the mutation operator is applied. Each recombined child is mutated exactly once.

SPEA2 is a recent multi-objective evolutionary optimizer introduced by Zitzler et al. [9]. SPEA2 maintains two sets of individuals: an archive that contains non-dominated individuals and a breeding population. In each generation, the two sets are merged and the fitness of the individuals is evaluated. The non-dominated individuals are then copied to the new archive. If the archive exceeds a predefined maximum size, SPEA2 applies a nearest neighbor density estimation technique to thin out clusters on the Pareto front. The fitness assigned to an individual considers thenumber of individuals it dominates (the dominance count), the number of individuals that are dominators (the dominance rank), and a density estimate based on the k-th nearest neighbor method. All individuals undergo a binary tournament selection which selects parents for recombination and mutation.

TSPEA2 is an algorithm we have devised in order to increase selection pressure on one objective while trying to keep diversity [6]. This should be beneficial for evolving circuits with a correctness property, where we will not be satisfied with a circuit unless the functional quality reaches a predefined level. Both SPEA2 and TSPEA2 use an archive and a breeding population and a selection scheme based on Pareto dominance ranking. TSPEA2, however, checks as a first selection rule in a binary tournament whether one of the two individuals dominates the other regarding the main objective. TSPEA2 has been motivated by an earlier algorithm, MO-Turtle GA presented by Trefzer et al. [10], that preferred a main objective over several other objectives during the evolution of analog circuits.

NSGAII was presented by Deb et al. in [11]. NSGAII separates the population into a hierarchy of Pareto fronts. The first level Pareto front is formed by the non-dominated individuals. These individuals are then removed from the population, and the second level Pareto front is formed by the now non-dominated individuals, and so on. A new elite population is filled by incrementally adding these Pareto fronts, starting with the level one front. In case the addition of the next level Pareto front exceeds the population's capacity, a density metric is used to select among the individuals of that front. A breeding population is created by using a standard GA scheme. Here, the selection operator takes the hierarchical Pareto front information and the density metric into account to achieve both diversity and a minimal distance to the optimal Pareto front.

μGA follows the original idea of Goldberg [12] who observed that a small number of individuals in a population is often sufficient for a converging optimization process. Consequently, he suggested an optimization scheme where a GA operates on a very small population. The situation in which all individuals have similar chromosomes is called nominal convergence. If such a nominal convergence is reached, the search process is relaxed by inserting randomly initialized individuals into the population. In [13], Coello Coello and Pulido combined the idea of Goldberg with the Pareto front diversity technique of Knowles and Corne [14]. Their μGA algorithm relies on three populations: an external archive population which contains non-dominated individuals of high diversity, the population memory which corresponds to the classical GA breeding population, and a non-replaceable population

which carries arbitrarily initialized individuals for the case of nominal convergence. In each step, a standard GA is applied on a small set of randomly selected individuals from the breeding and the non-replaceable population. After reaching nominal convergence, the best individuals are copied to the breeding and the external population. After several iterations of this scheme, a part of the breeding population is replaced by non-dominated individuals from the external population.

4 Experiments and Results

We have applied the different evolutionary optimizers to the following commonly used benchmarks for evolving digital circuits [15, 16, 5]: the 6 and 7 even parity function, $2+2$ and $3+3$ adders, and 2×2 and 3×3 multipliers. For the experiments, we have configured the CGP model as a single line of two-input gates (nodes). For the 6-parity function, the chromosome consists of 12 nodes, for the 7-parity function of 15 nodes, for the $2+2$ adder and 2×2 multiplier of 50 nodes, and for the $3+3$ adder and 3×3 multiplier of 200 nodes. As for the reference GA, the MOEAs rely on a two-point crossover with a recombination probability of 0.9. In each new individual, a single gene is mutated by modifying either the logic function or an input connection. The function set for the nodes is not restricted for the adder and multiplier experiments, i.e., the node function can be an arbitrary function of two inputs. For the parity experiments, however, the node function set has been restricted to `AND`, `NAND`, `OR` and `NOR`. Particularly, the `XOR` logic function is excluded, as otherwise the evolution of correct parity functions is not a challenge. All experiments have been conducted using the MOVES framework [17] for multi-objective evolutionary optimization of digital circuits.

As an example result, Figure 2 displays the development of the average functional quality for the 2×2 multiplier circuit. For this test problem, TSPEA2 shows the fastest convergence, followed by NSGAII, GA, SPEA2, and μGA. This result clearly shows that some multi-objective optimizers outperform the standard single-objective GA in evolving functionally correct circuits.

As we are interested in the asymptotical behavior of the algorithms regarding the functional quality of the evolved circuits, we have conducted several optimization runs for each test problem. We have stopped an optimization run when a correct circuit has been evolved. Otherwise, we have stopped the evolution after a predefined number of fitness evaluations. For the parity function this limit has been set to $14\cdot 10^6$ fitness evaluations, for the 3×3 multiplier to $6\cdot 10^6$ fitness evaluations, and for all other experiments to $20\cdot 10^6$ fitness evaluations.

We use two metrics to compare the algorithms. The first is the number of successfully evolved circuits among all runs of an experiment. This metric relates to robustness. The second metric is the computational effort as defined by Koza in [18] and can only be determined if a sufficient number of experiment runs produces correct circuits. In each run the optimization goal, i.e., the evolution of a functionally correct circuit, will be reached by some generation i. Having M fitness evaluations

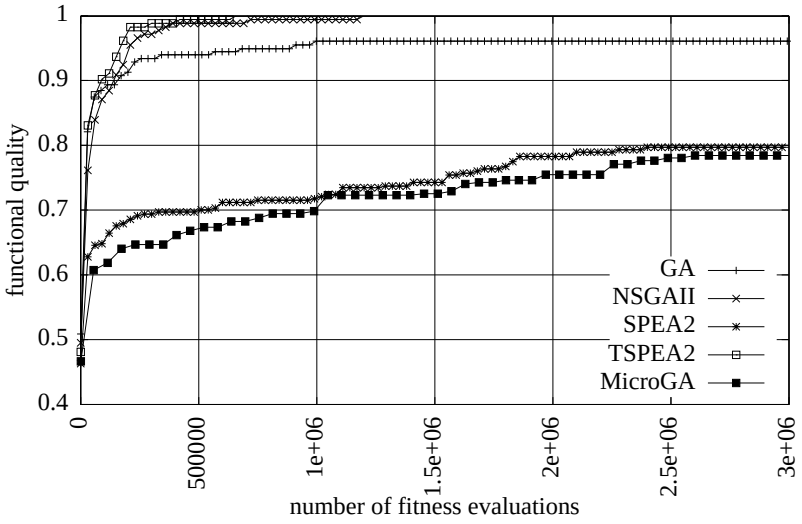

Fig. 2 Average functional quality of the best individuals over the number of fitness evaluations for the 2×2 multiplier (10 experiments runs)

per generation, the probability of reaching the optimization goal by generation i can then be expressed as follows:

$$P(M,i) = (\#\text{succeeded runs by generation } i)/(\#\text{runs})$$

From that we can determine $R(z)$, the number of independent runs that have to be conducted to reach the optimization goal with a certain probability z:

$$R(z) = \lceil \log(1-z)/\log(1-P(M,i)) \rceil$$

The estimated overall number of fitness evaluations required to reach the goal with probability z is then set to:

$$I(M,I,z) = M \cdot (i+1) \cdot R(z)$$

For each experiment with given M and z, the minimal value for $I(M,i,z)$ is determined as the computational effort of the experiment. In our experiments, we have set z to 99%.

The complete set of results is presented in Table 1. This table shows the computational effort and the number of successfully evolved circuits for 10 experiment runs for each test problem. SPEA2 and μGA did not succeed in evolving a sufficient number of correct circuits within the predefined number of fitness evaluations. Therefore, we did not compute the computational effort for these optimizers. The ranking of the algorithms with respect to the computational effort is shown in Table

2, where bold values indicate that the optimizers were able to evolve a functionally correct circuit in each single experiment run.

Table 1 Computational effort and number of correctly evolved circuits for standard GA and the MOEAs. The computational effort is given in multiples of 10^6. SPEA2 and μGA could not evolve a sufficient number of correct circuits to determine the computational effort.

	6-parity	7-parity	$2+2$ add	$3+3$ add	2×2 mult	3×3 mult
GA	0.09 / 10	0.25 / 10	0.09 / 10	6.63 / 9	0.79 / 8	– / 5
TSPEA2	0.15 / 10	2.02 / 10	1.42 / 10	1.55 / 8	0.59 / 10	1.89 / 10
NSGAII	1.14 / 10	3.65 / 10	1.10 / 10	3.61 / 10	1.04 / 10	3.29 / 9
SPEA2	– / 2	– / 0	– / 1	– / 0	– / 0	– / 0
μGA	– / 1	– / 0	– / 6	– / 2	– / 7	– / 0

From the results, we observe that the simpler functions, i.e., parity and $2+2$ adder, are easily evolved by the GA, and also by TSPEA2 and NSGAII. However, the multi-objective optimizers TSPEA2 and NSGAII require considerably more effort to evolve correct circuits. For the $3+3$ adder and the multipliers, the GA could not compete with TSPEA2 and NSGAII either in computational effort, the number of successfully evolved circuits, or both. The results indicate that with rising benchmark complexity, evolving a diverse population with regard to objectives such as circuit speed and area yields an improved robustness.

Table 2 Computational effort ranking. Bold values indicate experiments where every run produced a functionally correct circuit.

	6-parity	7-parity	$2+2$ add	$3+3$ add	2×2 mult	3×3 mult
GA	**1**	**1**	**1**	3	2	3
TSPEA2	**2**	**2**	**3**	1	**1**	**1**
NSGAII	**3**	**3**	**2**	**2**	**3**	2
SPEA2	4	4	5	5	5	4
μGA	5	4	4	4	4	4

5 Conclusion

In this paper, we have presented an experimental comparison of several multi-objective evolutionary optimizers and a standard genetic algorithm for the evolution of digital circuits. The goal was to investigate whether optimizing for circuit speed and area, besides functional quality, can improve the speed of convergence and robustness. We consider robustness a parameter of prime importance, especially for

self-optimizing autonomous systems that continuously run the evolutionary optimization process.

Our experimental results show that, for more complex benchmark problems, the classic genetic algorithm is indeed outperformed by two multi-objective optimizers, TSPEA2 and NSGAII. Two further optimizers, SPEA2 and μGA did not perform well for this task. In future, we plan to look at other secondary objectives to improve convergence and robustness. For example, there might be circuit properties besides area and speed that should be enforced. As scalability is one of the main challenges in evolvable hardware, the identification of suitable objectives for a *scalability-driven* evolution is of utmost importance.

Acknowledgment

This work was supported by the German Research Foundation under project number PL 471/1-2 within the priority program *Organic Computing*.

References

1. Sekanina, L.: Evolvable Components. Natural Computing Series. Springer (2004)
2. de Garis, H.: Evolvable Hardware – Genetic Programming of a Darwin Machine. In: Proceedings International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA), Springer (1993)
3. Higuchi, T., Niwa, T., Tanaka, T., Iba, H., de Garis, H., Furuya, T.: Evolving Hardware with Genetic Learning: A First Step Towards Building a Darwin Machine. In: Proceedings 2nd International Conference on Simulation of Adaptive Behavior (SAB), MIT Press (1993) 417–424
4. Coello Coello, C.A., Aguirre, A.H., Buckles, B.P.: Evolutionary Multiobjective Design of Combinational Logic Circuits. In: Proceedings of the 2nd NASA/DoD Workshop on Evolvable Hardware (EH), Los Alamitos, California, IEEE Computer Society (2000) 161–170
5. Kalganova, T., Miller, J.: Evolving More Efficient Digital Circuits by Allowing Circuit Layout Evolution and Multi-Objective Fitness. In: Proceedings of the 1st NASA/DoD Workshop on Evolvable Hardware (EH), Pasadena, California, IEEE Computer Society (1999) 54–63
6. Kaufmann, P., Platzner, M.: Toward Self-adaptive Embedded Systems: Multi-objective Hardware Evolution. In: Proceedings of the 20th International Conference on Architecture of Computing Systems (ARCS). Volume 4415 of LNCS., Springer (2007) 199–208
7. Miller, J.F., Thomson, P.: Cartesian Genetic Programming. In: Proceedings of the European Conference on Genetic Programming (ECGP), Springer-Verlag (2000) 121–132
8. Coello Coello, C.A., Aguirre, A.H.: Design of Combinational Logic Circuits through an Evolutionary Multiobjective Optimization Approach. In: Artificial Intelligence for Engineering Design, Analysis and Manufacturing. Volume 16., Cambridge University Press (2002) 39–53
9. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland (2001)
10. Trefzer, M., Langeheine, J., Meier, K., Schemmel, J.: Operational Amplifiers: An Example for Multi-objective Optimization on an Analog Evolvable Hardware Platform. In: International Conference on Evolvable Systems (ICES), Springer (2005) 86–97

11. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimisation: NSGA-II. In: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature (PPSN), Springer (2000) 849–858
12. Goldberg, D.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley (1989)
13. Coello Coello, C.A., Pulido, G.T.: A Micro-Genetic Algorithm for Multiobjective Optimization. In: First International Conference on Evolutionary Multi-Criterion Optimization (EMO). Volume 1993 of LNCS., Springer (2001) 126–140
14. Knowles, J.D., Corne, D.W.: Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. In: Evolutionary Computation. Volume 8., MIT Press (2000) 149–172
15. Miller, J.F.: 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 (GECCO). Volume 2., Morgan Kaufmann (1999) 1135–1142
16. Miller, J.F., Thomson, P., Fogarty, T.: Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study. In: Genetic Algorithms and Evolution Strategy in Engineering and Computer Science. John Wiley and Sons (1998) 105–131
17. Kaufmann, P., Platzner, M.: MOVES: A Modular Framework for Hardware Evolution. In: Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS), IEEE (5-8 Aug. 2007) 447–454
18. Koza, J.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)