Evolutionary Computing in Approximate Circuit Design and Optimization

Lukas Sekanina and Zdenek Vasicek
Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence
Bozetechova 2, 61266 Brno, Czech Republic
Email: sekanina@fit.vutbr.cz, vasicek@fit.vutbr.cz

***Abstract*—In this paper, we survey the methods that have been proposed to functional approximation of digital circuits. The main focus is on evolutionary computing, particularly Cartesian genetic programming (CGP), which can provide, in an automated manner, very good compromises between key circuit parameters. This is demonstrated in a case study – evolutionary approximation of an 8-bit multiplier.**

I. Introduction

Approximate computing is a modern paradigm which could bring new ideas to the design of low-power and high-performance computer-based systems [1], [2]. By means of smart approximations that can be introduced at various levels of the computer system (circuit, component, architecture, software, operating system, compiler) one can obtain significant benefits in terms of performance or power consumption for an acceptable error in data processing. Approximate computing exploits the fact that many applications are error resilient and the errors in computing are thus either invisible or acceptable. Approximate computing is also very relevant to *underdesigned and opportunistic computing* which attempts to explore the possibility of constructing machines naturally exploiting various imperfections of hardware [3]. This concept reflects the fact that after several decades of continual scalability of the manufacturing process, material properties of the newest chips started to negatively and significantly influence chips' properties in terms of intra-die and die-to-die variations, fluctuations in power consumption and maximum frequency, susceptibility to errors, and yield. In order to mitigate these undesired behaviors, redundant components and other reliability-strengthening mechanisms have to be introduced to circuit design and fabrication flows in various ways. On the other hand, approximate computing paradigm could naturally exploit these properties in order to build efficient low power and high-performance systems. As current CAD tools do not count in this situation, a new family of CAD tools is anticipated which will inherently support design, simulation, optimization and verification of underdesigned and approximate systems.

This paper deals with *approximate circuits* which can be defined as circuits relaxing the requirement on functional equivalence between the specification and implementation in order to reduce the area on a chip, delay, energy consumption or other measures that are important for users. Current CAD tools typically contain various heuristic algorithms attempting to quickly solve hard computational problems (Boolean satisfiability, partitioning, routing etc.) in which the circuit synthesis and optimization problems are often transformed to.

The last two decades witnessed a development of *bio-inspired circuit design* methods such as evolutionary design and evolvable hardware [4]. Utilizing the evolutionary design techniques has led to novel circuit implementations that are quite beyond the scope of designs reachable by conventional CAD tools [5], [6]. For example, an evolutionary logic optimization method enabled a 25% gate reduction (on average) in comparison with commercial and academia CAD tools [7]. In the case of evolution of approximate circuits, several studies showed that the computational effort is not as high as for conventional circuits, mainly because the specification can be violated and hence it is then much easier to find a suitable approximate solution [8]. It seems that the design of approximate circuits is in principle more suitable for the evolutionary approach than the design of conventional circuits and it would be very useful to introduce systematic design methods for approximate circuits based on the principles of evolutionary design. Moreover, extracting new (generally applicable) principles of approximate circuit design from evolved circuits could deepen our understanding to approximate computing.

The approximate circuit design problem can be formulated as a *multi-objective* optimization problem in which the accuracy, area, delay (or performance) and power consumption are conflicting design objectives. Designers expect obtaining a set of solutions which exhibit various trade-offs among key circuit parameters. The optimal set of such solutions is the so-called Pareto optimal front [9]. Current tools (such as [10], [11], [12]) typically approximate Pareto front by multiple executions of approximation engines which are initialized using different parameters.

The goal of this paper is to introduce several methods based on evolutionary computing (particularly, Cartesian genetic programming – CGP) that can be utilized to approximate circuit designs. It is expected that by means of these methods one can obtain significantly better designs in one run of the optimization algorithm than currently used single-objective optimization methods can achieve.

The reasons for using an advanced evolutionary approach (contrasted to a simple greedy search employed, for example, in [12]) are that the population-based approach suits well in finding multiple solutions and its niche-preservation methods can be exploited to discover diverse solutions. It is a well-

known fact that multiple applications of classic methods do not constitute an efficient parallel search [9].

In order to obtain a good approximate circuit, many candidate approximate circuits have to be generated and evaluated in the process of approximation. Hence the whole task requires a considerable computational time and the number of circuits that can be evaluated is the main constraint for the approximation engine. As the evolutionary algorithm generates and evaluates many candidate circuits, it is impossible to precisely measure parameters of all of them in the course of evolution. Hence the methods presented in this paper have to quickly estimate key circuit parameters (area and delay) during the optimization process. At the end of evolution, the resulting approximate circuits are implemented using a standard circuit design flow and compared with their accurate counterparts.

The rest of the paper is organized as follows. Section II deals with existing methods for approximate circuit design. The principles of evolutionary circuit design are surveyed in Section III. Section IV introduces three different approaches to circuit approximation based on CGP. A case study is presented in Section V. Conclusions are given in Section VI.

II. Approximate Circuits

In approximate circuits, the accuracy (or quality) of the output is traded for improvements in power consumption or performance. Approximate circuits are constructed by either modifying accurate (original) circuits or developing required circuits from scratch. There are two types of methods that can be utilized to approximate circuit designs. The first group exploits technology-oriented techniques such as voltage over scaling and over clocking. In the second group of techniques, referred to as *functional approximation*, the aim is to modify the original logic function of the circuit in order to reduce power consumption or increase performance (operation frequency). Functional approximation is often combined with voltage over scaling or over clocking.

However, there is not still a well-established methodology for automated construction of approximate systems and circuits which could provide a good trade-off among key parameters. A recent comprehensive survey [3] clearly states in its "Implications for Circuits and Architectures" section that

> Much research needs to be done to functionally or parametrically underdesign large general class of circuits automatically. Mechanisms to pass application intent to physical implementation flow (especially to logic synthesis in case of functional underdesign) need to be developed.

We will only focus on the principles of functional approximation in this paper. Manual redesign was the first approach to circuit approximations; see, for example, the approximate adders [13] and multipliers [14]. However, the manual redesign suffers from scalability and efficiency. Recently developed systematic methods have used problem-specific heuristics for automated selecting of subcircuits suitable for approximation, other heuristics for performing actual approximations, and various verification engines to prove that the chosen approximation satisfies the predefined quality constraints. These methods are *error-oriented* in the sense that a target error is specified (fixed) and remaining circuit parameters are optimized.

The Systematic methodology for Automatic Logic Synthesis of Approximate circuits (SALSA) starts with a description of the accurate circuit and an error constraint that specifies the type of error that can be accepted [10]. SALSA introduces the quality function which takes the outputs from both the original circuit and approximate circuit and decides if the quality constraints are satisfied. The quality function outputs a single Boolean value. The SALSA algorithm attempts to modify the approximate circuit with the goal of keeping the output of the quality unchanged. The concept of approximation by means of the quality function has been extended to sequential circuits in [15].

Another method, SASIMI, tries to identify signal pairs in the circuit that exhibit the same value with a high probability, and substitutes one for the other [11]. These substitutions introduce functional approximations. Unused logic can be eliminated from the circuit which results in area and power savings.

ABACUS creates an abstract synthesis tree (AST) from the input behavioral description and then applies various operators to the AST using an iterative stochastic greedy approach [12]. Candidate designs are evaluated in terms of accuracy, power consumption and area in a single objective optimization scenario. The objectives are combined together using a linear weigh function. The Pareto front is obtained from multiple runs of the search algorithm (about 50 candidate circuits are generated in each run).

The aforementioned methods try to approximate the Pareto optimal front by solutions obtained from multiple runs of single-objective approximation engines. However, in many cases, the resulting solutions do not cover the whole Pareto front and the design alternatives are centered around a few dominant design alternatives. These methods utilize the standard design flow to construct and evaluate every candidate solution, which is very time consuming. On the other hand, the circuit parameters obtained are very close to real ones.

An integrated approach to approximate computing has been developed in [16]. It consists in automatic resilience characterization of the target application in order to identify those parts of the application that are suitable for approximation. The application is then implemented using a specialized hardware (processor) whose components can be tuned in accordance with the desired output quality to adapt their energy consumption. In addition to the off-line tuning of the application, an automated on-line regulation of the degree of approximation is supported by the hardware.

III. Evolutionary Methods in Circuit Design

There are many studies demonstrating that evolutionary circuit design can produce efficient implementations of electronic circuits, for example, in domains of analog circuits [6], logic optimization [7], image filters [17], and classifiers [18]. We will briefly introduce Cartesian genetic programming and the reasons for its utilization in the circuit design and optimization.

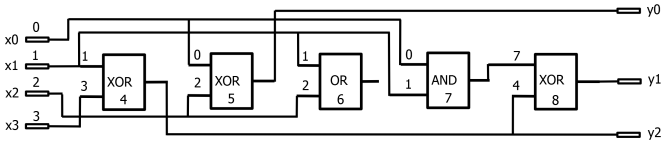

Fig. 1. A candidate circuit represented by CGP with parameters: $n_i = 4, n_o = 3$, $n_c = 5$, $n_r = 1$, $l = 4$, $\Gamma = \{0^{AND}, 1^{OR}, 2^{XOR}\}$. Chromosome: 1, 3, 2; 0, 2, 2; 1, 2, 1; 0, 1, 0; 7, 4, 2; 5, 8, 4.

A. Circuit Evolution Using Cartesian Genetic Programming

CGP enables to design electronic circuits from scratch or optimize structures and parameters of existing circuits [19]. In order to utilize CGP, one has to deal with the problem representation, genetic operators and fitness function.

A candidate circuit is modeled by means of a directed acyclic graph whose nodes (gates) are organized in n_c columns and n_r rows. Depending on a particular application, the nodes can be elementary logic functions, transistors or high-level components such as adders or multipliers.The connections are defined over b bits. The circuit utilizes n_i primary inputs and n_o primary outputs. Each node input can be connected either to the output of a node placed in previous l columns or to one of the primary circuit inputs, where l is one of CGP parameters.

The primary inputs and the outputs of the nodes are labeled $0, 1 \ldots n_c \cdot n_r + n_i - 1$ and considered as addresses which links can be connected to. A candidate solution consisting of two-input nodes is represented in the chromosome by $n_r \cdot n_c$ triplets (x_1, x_2, ψ) determining for each processing node its function ψ ($\psi \in \Gamma$), and addresses of nodes x_1 and x_2 which its inputs are connected to. The last part of the chromosome contains n_o integers specifying either the nodes where the primary outputs are connected to or logic constants ('0' and '1') which can directly be connected to the primary output. While the chromosome size is constant for a given product $n_r \cdot n_c$, the phenotype size is variable and measured as the number of used nodes (gates). See an example in Fig. 1.

CGP employs a (1 + λ) evolution strategy whose pseudo-code is given in Algorithm 1. The initial population Q of CGP is created either randomly or by means of existing circuits. The next step consists in the evaluation of candidate circuits using the fitness function. In the simplest case, circuits responses are calculated for a set of test vectors. The goal is to minimize the difference between the obtained vectors and desired vectors. Each member of Q then receives one fitness value and the highest-scored individual becomes a new parent of the next population. CGP solely employs a mutation operator, which randomly modifies up to h integers of the chromosome. The termination criterion depends on a particular application.

B. Multi-objective Search Algorithms

If a multi-objective optimization is conducted, there are several fitness functions formulated, each of them reflecting to what extent a given circuit parameter (accuracy, area, delay etc.) satisfies the specification. A straightforward approach to the multi-objective optimization is to convert the multi-objective problem to a single objective one using a weight function $\sum w_i f_i$, where w_i is the weight of the i-th objective. Because a single run of an optimizer which utilizes the sum yields only one solution, multiple runs are needed for obtaining various trade-offs. The proper setting of weights w_i is not an easy task and is usually based on the user intuition. Another limitation of the weight function lies in the fact that certain Pareto-optimal solutions are not reachable in the case of nonconvex objective space [9]. Since it is difficult to detect whether the resulting objective space is nonconvex, the weight function has to be applied with caution.

Algorithm 1: CGP

Input: CGP parameters, fitness function
Output: The highest scored individual p and its fitness

```
Q ← randomly generate parent p and its λ offspring;
EvaluatePopulation(Q);
while ⟨terminating condition not satisfied⟩ do
    α ← highest-scored-individual(Q);
    if fitness(α) ≥ fitness(p) then
        p ← α;
    Q ← create λ offspring of p using mutation;
    EvaluatePopulation(Q);
return p, fitness(p);
```

Truly multiobjective optimization algorithms operate in a different way. The key task is to compare two candidate solutions. In order to do so, Pareto-dominance relations are employed [9]: We say that solution $\vec{x}^{(1)}$ *dominates* another solution $\vec{x}^{(2)}$ if the following conditions are satisfied: (i.) The solution $\vec{x}^{(1)}$ is no worse than $\vec{x}^{(2)}$ in all objectives. (ii.) The solution $\vec{x}^{(1)}$ is strictly better than $\vec{x}^{(2)}$ in at least one objective.

The result of the multi-objective optimization is no longer a single solution, but a set of solutions. The non-dominated subset of all possible solutions is called Pareto-optimal set (front). In practice, the goal is to find a set of solutions as close as possible and as diverse as possible with respect to the Pareto-optimal front. Examples of truly multi-objective evolutionary algorithms are Vector Evaluated Genetic Algorithm (VEGA), Strength Pareto Evolutionary Algorithm (SPEA), and non-dominated sorting genetic algorithm (NSGA-II).

C. Why Evolutionary Approximation?

We consider the following properties of the evolutionary design method as essential in the context of circuit approximation:

- Employing evolutionary computation is natural with respect to its goal in the approximation task. Small modifications introduced in the progress of evolution to a population of circuits and the principle of the survival of the fittest naturally lead to discovering such circuits which show very good compromises between the error and area (power).

- The evolutionary design can produce and optimize partially working solutions even if sufficient resources are not available, which is not the case of conventional methods.
- Evolutionary algorithms can naturally handle multiple conflicting design objectives.
- In the fitness function, arbitrary error measures can be introduces, whereas existing methods typically utilize the worst error only.
- As many candidate circuits are evaluated, very efficient circuit designs, practically unreachable by means of conventional methods, which generate and analyze only several circuits, can be discovered.

The main disadvantage of the evolutionary design method is its enormous computational effort. Many candidate circuits have to be generated and evaluated. In the case of small circuits, responses are calculated for all possible assignments to the inputs and compared with desired values. As this approach is not scalable, functionality of complex circuits has to be evaluated using advanced techniques (e.g. [7]). It is not tractable to precisely evaluate circuit parameters (in particular, power consumption) using professional design tools in the course of evolution. Hence basic circuits characteristic are estimated in the fitness function and only resulting circuits are implemented using the standard design flow at the end of evolution. For example, it is expected that power consumption is highly correlated with the area and hence only the area is estimated in the fitness function.

Another objection against evolutionary design is that the method is not deterministic and it is difficult (or impossible) to understand the evolved solutions. However, these issues are almost irrelevant in the scope of approximate circuits because other automated methods are typically stochastic and it is, in principle, extremely difficult to understand the resulting approximations.

IV. CGP in Circuit Approximation

In the following paragraphs, we will present tree approaches to circuit approximation based on CGP.

A. Resources-oriented Method

This method exploits the fact that CGP can produce a partially working solution even if sufficient resources for constructing an exact circuit are not available. The idea is to evolve a circuit showing a minimum error using k_i components (gates) providing that $k_i < K$ and K is the number of components (gates) required to implement a correct circuit. CGP is considered as a single-objective method which is executed several times with different k_i as the parameter. It provides a set of approximate circuits, each of which typically exhibits different trade-off between the functionality and the number of gates. The main advantage is that the user can control the used area (and so power consumption) more comfortably than by means of the error-oriented methods.

The method was employed to approximate single-output benchmark combinational circuits [20], small adders (up to 4 bits) [20], small multipliers (up to 4 bits) [21], and 9-input and 25-input median circuits operating over 8 bits [21]. Larger multipliers were constructed from evolved smaller ones using the approach presented in [14].

B. Error-oriented Method

In paper [8], we proposed a complementary design approach. The user is supposed to define a required error level e_{max} (e.g. the average error magnitude). In the first step, CGP, which is seeded by a conventional fully functional implementation, is utilized to modify the seed in order to obtain a circuit with predefined e_{max}. After obtaining that circuit, in the second step, CGP can minimize the mean error, the number of gates or other criteria providing that e_{max} is left unchanged. Again, the method is single objective and multiple runs are required to construct the Pareto front. The method was evaluated in the task of 4-, 5-, 6-, 7- and 8 bit approximate multiplier design. The error-oriented approach tends to be less computationally demanding than the resources oriented method.

C. Multi-objective CGP

In the multi-objective method (MO), good compromises among three conflicting design objectives – error, area and delay – are together sought in a single run of CGP. The $1 + \lambda$ search strategy is replaced by procedures of NSGA-II which implement non-dominated sorting of the population (non-dominated solutions are emphasized) and diversity preservation mechanisms (less crowded points of the search space are promoted) – details can be found in [22]. The maximum allowed error E_{max} (which the designer is going to observe and accept in the resulting Pareto fronts) is defined as a constraint in the algorithm. If the fitness $f_{error} > E_{max}$, the solution is considered as unacceptable.

V. Case Study: Approximate 8-bit Multiplier

In the case study, we will compare a single-objective (SO) CGP (which utilizes weigh function for the objectives) and MO CGP in the task of 8-bit multiplier approximation. Both algorithms are seeded with a fully functional version of parallel carry save adder-based multiplier.

A. Estimation of Circuit Parameters

In order to estimate electrical parameters of a given circuit, the area and delay are calculated using the parameters defined in the liberty timing file available for a given semiconductor technology.

Delay t_d of a cell c_i is modeled as a function of its input transition time t_s and capacitive load c_l on the output of the cell, i.e. $t_d(c_i) = f(t_s^{c_i}, c_l^{c_i})$. Delay of a circuit C is determined as delay of the longest path:

$$Delay(C) = \max_{\forall p \in \text{path}} \sum_{c_i \in p} t_d(c_i).$$

The capacitive load on the circuit outputs is chosen to be equal to the input capacitance of an inverter cell. The transition

time on circuit inputs corresponds to the transition time on the output of an inverter cell.

The area of a circuit C is calculated as the sum of areas of all the cells c_i involved in the circuit:

$$Area(C) = \sum_{c_i \in C} area(c_i).$$

B. Handling the Accuracy

Various error criteria can be utilized to evaluate the quality of an approximate arithmetic circuit. The average error magnitude $E_{avg}(C)$ is employed in this paper. This metric is defined as the sum of absolute differences in magnitude between the original and approximate circuits averaged over all inputs:

$$E_{avg}(C) = \frac{\sum_{\forall i} |O(C_{orig}, i) - O(C, i)|}{2^{2w}}, \qquad (1)$$

where $O(C_{orig}, i)$ denotes the output value of the fully functional circuit for the input vector i, $O(C, i)$ denotes the output value of approximate circuit C and w specifies the bit-width.

Let E_{max} be the maximum acceptable average error. In the case of MO, solutions with the error greater than E_{max} are infeasible. In order to construct the Pareto front in the case of SO, CGP is executed with different target errors E_i, $E_i \leq E_{max}$. The fitness value $fitness_{L1}$ is calculated as the relative absolute difference from required error level and the goal is to minimize this difference, i.e.

$$fitness_{L1}(C) = \frac{|E_{avg}(C) - E_i|}{E_i}.$$

Once a circuit showing the target error ($fitness_{L1} < 0.01$) is obtained, a weight fitness function is employed. Each objective is normalized to the interval $< 0, 1 >$ and weighted with weights w_e, w_a and w_d ($w_e + w_a + w_d = 1$). Then,

$$fitness_{L2}(C) = \begin{cases} w_e E'_{avg}(C) + \\ w_a Area'(C) + \\ w_d Delay'(C), & \text{if } fitness_{L1} < 0.01 \\ \infty, & \text{otherwise,} \end{cases}$$

where the apostrophes denote normalized values with respect to the original, fully functional circuit.

C. Experimental Setup

In both approaches we utilized the following CGP parameters: $k = 5$, $l = n_c = 320$ (it is the number of gates in the original fully functional multiplier), $n_r = 1$, $b = 1$, $n_i = 16$, $n_o = 16$, and $E_{max} = 2.5\%$. While MO is executed with E_{max}, SO is executed SO_{run} times; one run for one error level from 0% do 2.5%. The weights for SO are $w_e = 0.12$, $w_a = 0.5$, $w_d = 0.38$ (it is the most efficient setting that we found).

In order to fairly compare SO and MO, both methods will generate and evaluate the same number of candidate circuits. We performed 5, 25, 50, 250 and 500 thousand evaluations (fitness calls). For example, 5000 evaluations correspond with 100 generations of MO ($\lambda_{MO} = 50$). In the case of SO, 50 generations are produced for each of 20 error levels ($\lambda_{SO} = 5$, $SO_{run} = 20$). This number of evaluations is very low from the perspective of evolutionary circuit design; however, this number is still much higher than in conventional methods [12].

The experiments were conducted for I3T25 technology (0.35 μm digital process). The following cells are considered: *and, or, xor, nand, nor, xnor, buf, inv*, with corresponding relative areas 1.333, 1.333, 2, 1, 1, 2, 1.333, and 0.667.

D. Properties of Evolved Multipliers

The best-obtained Pareto fronts (projected to two 2D graphs) are given for each particular number of evaluations in Fig. 2 (25 independent runs performed). When 5000 evaluations are allowed, MO performs better than SO. However, SO becomes more efficient if more evaluations can be used. The improvement of the SO approach is probably due the fact that the error is fixed and the overall effort can be put into minimizing the area and delay. On the other hand, MO has to cover the whole Pareto front and the available time seems to be insufficient to compete with SO. In terms of performance, while 240 evaluations/s can be done using MO, the SO approach enabled 284 evaluations/s on a common PC.

Some of evolved circuits were implemented using a standard design flow by Cadence Encounter RTL Compiler. Table I shows that our estimated values are almost perfectly matched with the results of the professional design tool (compare the percentages in the parentheses which are related to original circuits showing a 0% error).

A direct and fair comparison with the literature is not possible because the authors have used different implementations of accurate multipliers and circuit fabrication technology. The following results were taken from the literature for the 8-bit multiplier: SASIMI [11] reports 45% power reduction with 0.5% average error, Kulkarni et al. [14] reports 33% power reduction with 3.25% average error and Gupta et al. [13] reports 35% power reduction with 2.5% average error. The proposed method led to 71% power reduction with 0.6% average error, which seems to be a very good result.

TABLE I
PARAMETERS OF THE APPROXIMATE 8-BIT MULTIPLIERS. PERCENTAGES IN THE PARENTHESES ARE RELATED TO THE ORIGINAL CIRCUITS SHOWING A 0% ERROR.

	estimated			professional tool		
error	delay [ns]		rel. area	delay [ns]	area [μm^2]	pwr [μW]
0.0%	13.1 (100%)	495	(100%)	12.1 (100%)	24245 (100%)	1367 (100%)
0.6%	8.7 (66%)	175	(35%)	8.0 (65%)	8480 (34%)	409 (29%)
1.3%	6.3 (48%)	118	(23%)	5.6 (46%)	5424 (22%)	233 (17%)
1.9%	5.4 (41%)	92	(18%)	5.1 (42%)	4513 (18%)	164 (12%)
2.5%	4.5 (34%)	64	(13%)	4.3 (35%)	3118 (12%)	106 (7%)

VI. CONCLUSIONS

In this paper, we surveyed methods that have been proposed to approximate circuit designs. In particular, we have dealt with an evolutionary approach to functional approximation based on CGP. Three different techniques were presented (error-oriented, resources-oriented and truly multi-objective).

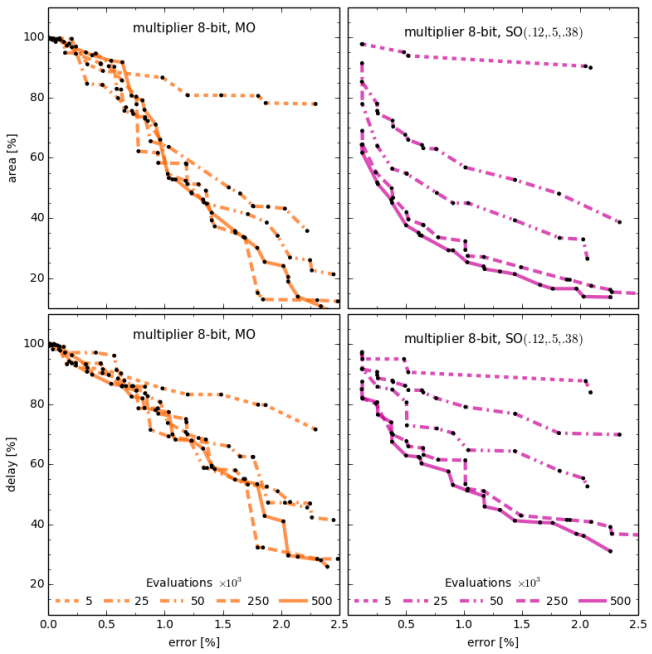

Fig. 2. Resulting Pareto fronts with respect to the number of evaluations

In all cases, parameters of candidate circuits were estimated in the fitness function. Resulting circuits were synthesized by means of a professional design tool and their parameters compared with estimated values. Although the multi-objective approach is intuitively the most useful one, we demonstrated in the case study that a single-objective approach can provide better results when a sufficient number of evaluations is enabled.

It seems that the evolutionary approach can generate, in an automated manner, very good compromises between key circuit parameters. The main drawback is its limited scalability for arithmetic circuits. Components of multimedia and other circuits (such as median circuits), in which training data sets are processed, can be approximated by CGP even if they are relatively complex [21].

Our future research in the evolutionary circuit approximation will primarily be devoted to eliminating the scalability issues. The principles of evolutionary approximation should also be introduced to other layers of the system stack. Finally, benchmark circuits are needed to allow a fair comparison of circuit approximation methods and tools.

VII. Acknowledgments

This work was supported by the Czech science foundation project 14-04197S and the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070.

References

[1] S. T. Chakradhar and A. Raghunathan, "Best-effort computing: Rethinking parallel software and hardware," in *Proceedings of the 47th Design Automation Conference – DAC*. ACM, 2010, pp. 865–870.

[2] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Proc. of the 18th IEEE European Test Symposium*. IEEE, 2013, pp. 1–6.

[3] P. Gupta, Y. Agarwal, L. Dolecek, N. Dutt, R. K. Gupta, R. Kumar, S. Mitra, A. Nicolau, T. S. Rosing, M. B. Srivastava, S. Swanson, and D. Sylvester, "Underdesigned and opportunistic computing in presence of hardware variability," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 8–23, 2013.

[4] P. C. Haddow and A. M. Tyrrell, "Challenges of evolvable hardware: past, present and the path to a promising future," *Genetic Programming and Evolvable Machines*, vol. 12, no. 3, pp. 183–215, 2011.

[5] J. R. Koza, "Human-competitive results produced by genetic programming," *Genetic Programming and Evolvable Machines*, vol. 11, no. 3–4, pp. 251–284, 2010.

[6] T. McConaghy, , P. Palmers, M., and G. Gielen, "Trustworthy genetic programming-based synthesis of analog circuit topologies using hierarchical domain-specific building blocks," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 4, pp. 557–570, 2011.

[7] Z. Vasicek and L. Sekanina, "A global postsynthesis optimization method for combinational circuits," in *Proc. of the Design, Automation and Test in Europe, DATE*. EDA Consortium, 2011, pp. 1525–1528.

[8] ——, "Evolutionary design of approximate multipliers under different error metrics," in *IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems 2013*. IEEE, 2014, pp. 135–140.

[9] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley, 2001.

[10] S. Venkataramani, A. Sabne, V. J. Kozhikkottu, K. Roy, and A. Raghunathan, "Salsa: systematic logic synthesis of approximate circuits," in *The 49th Annual Design Automation Conference 2012, DAC '12*. ACM, 2012, pp. 796–801.

[11] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: a unified design paradigm for approximate and quality configurable circuits," in *Design, Automation and Test in Europe, DATE'13*. EDA Consortium San Jose, CA, USA, 2013, pp. 1367–1372.

[12] K. Nepal, Y. Li, R. I. Bahar, and S. Reda, "Abacus: A technique for automated behavioral synthesis of approximate computing circuits," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '14. EDA Consortium, 2014, pp. 1–6.

[13] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 124–137, 2013.

[14] P. Kulkarni, P. Gupta, and M. D. Ercegovac, "Trading accuracy for power in a multiplier architecture," *J. Low Power Electronics*, vol. 7, no. 4, pp. 490–501, 2011.

[15] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan, "Aslan: Synthesis of approximate sequential circuits," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '14. EDA Consortium, 2014, pp. 1–6.

[16] V. Chippa, S. Venkataramani, S. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing: An integrated hardware approach," in *2013 Asilomar Conference on Signals, Systems and Computers*. IEEE, 2013, pp. 111–117.

[17] Z. Vasicek, M. Bidlo, and L. Sekanina, "Evolution of efficient real-time non-linear image filters for fpgas," *Soft Computing*, vol. 17, no. 11, pp. 2163–2180, 2013.

[18] P. Kaufmann, K. Glette, T. Gruber, M. Platzner, J. Torresen, and B. Sick, "Classification of electromyographic signals: Comparing evolvable hardware to conventional classifiers," *IEEE Tran. Evolutionary Computation*, vol. 17, no. 1, pp. 46–63, 2013.

[19] J. F. Miller, *Cartesian Genetic Programming*. Springer-Verlag, 2011.

[20] L. Sekanina and Z. Vasicek, "Approximate circuits by means of evolvable hardware," in *2013 IEEE International Conference on Evolvable Systems*, ser. Proceedings of the 2013 IEEE Symposium Series on Computational Intelligence (SSCI). IEEE CIS, 2013, pp. 21–28.

[21] Z. Vasicek and L. Sekanina, "Evolutionary approach to approximate digital circuits design," *IEEE Tran. on Evolutionary Computation – to appear*, pp. 1–13, 2015.

[22] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.