Evolving Component Library for Approximate High Level Synthesis

Filip Vaverka, Radek Hrbacek and Lukas Sekanina
Brno University of Technology, Faculty of Information Technology
IT4Innovations Centre of Excellence, Brno, Czech Republic
Email: ivaverka@fit.vutbr.cz, ihrbacek@fit.vutbr.cz, sekanina@fit.vutbr.cz

***Abstract*—An approximate computing approach has recently been introduced for high level circuit synthesis (HLS) in order to make good use of approximate circuits at system and block level. It is assumed in HLS algorithms that a component library containing various implementations of elementary circuit components is available. An open problem is how to construct such a component library in the context of approximate computing, where the component's error is a new design variable and hence many compromise implementations exist for a given component. In this paper, we first introduce a multi-objective Cartesian genetic programming method to create a comprehensive component library containing hundreds of Pareto optimal implementations of approximate 8-bit adders and multipliers, where the error, area and delay are simultaneously optimized. Another multi-objective evolutionary algorithm is employed to solve the so called binding problem of HLS, in which suitable approximate components are assigned to nodes of the data flow graph describing a complex digital circuit. Two approaches are then proposed and compared in order to reduce the size of the library of approximate components. It is shown that a random sub-sampling of the component library provides satisfactory results in the context of our study. The proposed methods are evaluated using two benchmark circuits – the reduce (sum) and DCT circuits.**

I. Introduction

In modern circuit design tools, the circuit complexity is handled by combining many approaches including, among others, decomposition and abstraction. Circuits are represented in different ways at different levels of abstraction. For example, the requested functionality is specified in a common programming language such as C. By means of *high-level synthesis* (HLS), a register transfer (RT) level representation is created from the C code. Resulting circuit is then implemented using components available in the component library or synthesized by general-purpose logic synthesis methods. The component library contains frequently-used components (such as adders, multipliers, multiplexers, etc.) that are carefully optimized for a given fabrication technology. Each component is typically available in several instances showing different parameters, for example, fast but area-demanding vs. slow but area-saving multipliers. The problem is formalized in such a way that a circuit meeting a given latency constraint is sought while its implementation cost (the area on a chip) has to be minimized.

Recent years have witnessed a rapid development in *approximate computing* [1]. As many applications are inherently error-resilient, it is highly requested to exchange this resilience for improvements in circuit parameters, primarily, in power consumption reduction. In order to approximate digital circuits, manual as well as automated circuit approximation methods have been developed. These methods typically provide compromise circuit implementations along a Pareto front showing different trade-offs between key circuit parameters (delay, area, power consumption) and error. However, the circuit approximation methods exhibit a kind of scalability problem as only relatively simple circuits have been approximated so far. One of the reasons is that calculating the *exact* error (i.e. evaluating a complex candidate approximate circuit for the whole input range) is difficult and so time consuming under common error metrics such as the average error.

In order to approximate complex circuits, the high level synthesis methodology known for common circuits has been adopted for approximate circuits [2]. In this approach, the component library contains approximate implementations of components which differ not only in standard circuit parameters, but also in the accuracy. The overall objective is usually to minimize total leakage energy consumption while latency and accuracy constraints have to be satisfied.

In our previous work, we developed several evolutionary algorithm (EA)-based circuit approximation methods [3], [4], [5]. Regarding the complexity of evolved approximate circuits, they could be considered as typical approximate components of the library used in HLS (e.g., 8-bit adders, 8-bit multipliers). As the proposed circuit approximation methods are fully automated and accelerated, arbitrary combinational circuits (up to a certain complexity) can be approximated meeting a given error constraint. Moreover, different types of error metrics can be employed. A very rich component library containing hundreds of approximate versions of a given circuit can thus be generated.

It is an open problem how to compose a library containing approximate components for a particular HLS task. The aim of this paper is to investigate the impact of the component library size and construction on the quality of approximate circuits produced by HLS. As this is the first study in this direction, we consider only one step of HLS – the *binding* which assigns operations of the algorithm to specific instances of components from the library.

Cartesian genetic programming (CGP) is connected with the NSGA-II algorithm [6] to form a multi-objective evolutionary approximate circuit design tool. This tool is utilized to generate a component library, i.e. many instances of approximate adders and multipliers showing different parameters in terms

of area, delay and error. In total, the resulting component library contains 473 approximate 8-bit adders and 500 approximate 8-bit multipliers, i.e. significantly more components than usually used in HLS.

Once the library containing approximate components is available, it is utilized in the binding task of HLS. The input is a scheduled data flow graph (DFG) representing the operations that have to be executed and the total latency. The binding problem is solved using NSGA-II which assigns components of the library to nodes of DFG. The resulting assignments are presented in Pareto fronts to show various trade-offs among the circuit delay, area and error. The binding optimization based on NSGA-II is repeated with component libraries constructed with different constrains and parameters in order to find out the impact of the component library size (and other parameters) on the resulting approximate circuits. For each candidate binding, it is necessary to calculate the total error of the circuit. As the circuits are complex and computing the exact error is intractable, the total error is estimated using circuit simulation and a method proposed in [2]. The impact of the component library construction on the overall approximation process is evaluated using two benchmarks (8-to-1 reduction and 8-input fixed-point discrete cosine transform (DCT)).

The rest of the paper is organized as follows. Section II briefly surveys relevant state of the art in the areas of HLS, approximate computing and evolutionary circuit approximation. In Section III, a multi-objective CGP is introduced and used to evolve approximate implementations of components of the library. The evolved component library is then employed in the binding task whose results are reported in Section IV. Conclusions are given in Section V.

II. Related Work

A. High Level Synthesis

HLS is an algorithmic approach introduced for an efficient design and implementation of complex digital circuits. HLS synthesis raises the design abstraction level and allows rapid generation of optimized RT level hardware for performance, area, and power requirements. HLS has significantly been improved in recent years [7], mainly due improving the algorithms behind its elementary steps.

Starting from the high-level description of an application, an RT level component library and specific design constraints, an HLS tool executes the following tasks [8]: compiles the specification, allocates hardware resources (functional units, storage components, buses, and so on), schedules the operations to clock cycles, binds the operations to functional units, binds variables to storage elements, binds transfers to buses, and generates the RT level architecture.

A key structure behind HLS is a data flow graph, in which the nodes represent operations and the connections between the nodes represent data dependencies and indicate the order of operations. Allocation determines the type and the number of resources needed to satisfy the design constraints. All operations of DFG must be scheduled into cycles. Every operation must be bound to one of the functional units (components) capable of executing the operation. If there are several units available, the binding algorithm must optimize this selection. Similarly, variables and connections must be bound to corresponding resources. Any component is available in the component library in several instances that have different area/delay/power trade-offs. A data path state machine is created to control the scheduled design.

B. Approximate Computing

Approximate computing exploits the gap between the level of accuracy required by the applications/users and that provided by the computing system for achieving diverse optimizations [1]. Two major approaches can be traced in the literature.

A *bottom-up* approach exploits the fact that the exact computing utilizing nanometer transistors provided by recent technology nodes is extremely expensive in terms of energy requirements and reliable behavior. An open question is how to effectively and reliably compute with a huge amount of unreliable components. A possible solution could allow performing of imprecise computations by the unreliable components without introducing common fault tolerant mechanisms [9]. If the resulting error were acceptable, the benefits would be in obtaining very energy efficient operations.

A *top-down* approach is based on simplifying correctly working hardware and software components that are employed in application domains such as multimedia, graphics, data mining, and big data processing. These applications are inherently *error resilient*. This resilience can be exchanged for improvements in power consumption, throughput or implementation cost. One of the approximation techniques is *functional approximation* whose principle is to implement a slightly different function to the original one provided that the error is acceptable and key system parameters are improved. Various tools now perform the functional approximation which can further be combined with voltage over scaling to improve energy efficiency of resulting circuits [10], [11], [12].

Following the top-down scenario, an approximate HLS has recently been introduced for complex digital systems. Li et al. introduced a library of approximate components that were used in the scheduling and component allocation/binding for data intensive error-resilient applications [2]. Moreover, a variance-based error model was proposed to evaluate candidate solutions (see Section IV-C).

C. CGP and Circuit Approximation

1) CGP: In CGP, candidate solutions are represented in a two-dimensional array of programmable nodes [13]. An n_{i}-input and n_{o}-output combinational circuit is modeled using an array of $n_{\mathrm{c}} \cdot n_{\mathrm{r}}$ programmable nodes forming a Cartesian grid. A set of available n_{a}-input node functions is denoted Γ. The levels-back parameter l constraints which columns a node can get its inputs from. No feedback is allowed in the basic version of CGP. The primary inputs and programmable nodes are uniquely numbered. For each node the chromosome contains (n_{a}+1) values that represent (i) the node function and (ii) n_{a} addresses specifying the input connections. The chromosome

also contains n_o values specifying the gates connected to the primary outputs. The chromosome size is $n_c n_r (n_a + 1) + n_o$ integers. The search algorithm utilized by CGP is a simple mutation-based $(1 + \lambda)$ search strategy.

2) Evolutionary Circuit Approximation: CGP can naturally be extended for circuit approximation because the fitness function always includes a component measuring the circuit functionality. If the circuit under approximation is not complex, it is possible to evaluate its responses for all possible input combinations and compute its exact error according to an arbitrary chosen error metric. In the case of more complex circuits, candidate circuits are evaluated using a training set (i.e. a subset of all possible vectors) and the resulting error is thus only estimated (see approximate median filters in [3]). If an exact error is requested, a formal relaxed equivalence checking has to be employed as demonstrated for relatively complex combinational circuits in [5], [14]. However, these formal methods are currently available only for a very restricted set of error metrics.

The CGP-based approximation methods can be classified as:

- *Error-oriented*, in which CGP tries to evolve a circuit showing a predefined error, and consequently, to optimize circuit parameters without worsening this error [5].
- *Resources-oriented*, in which resources (e.g. the number of gates) are constrained and CGP is used to minimize the circuit error with available resources [3].
- *Multi-objective*, in which all criteria are optimized together using a multi-objective EA such as NSGA-II [4].

III. Evolutionary Design of Approximate Components

Recently, the evolutionary approach was applied in the task of approximate circuits design with respect to multiple objectives and conventional circuits were used as an initial population [4]. The method is based on a multi-objective CGP implementation inspired by the NSGA-II algorithm. For a given set of conventional circuits, the method is able to produce a (larger) set of Pareto optimal solutions in terms of the error, power consumption and delay. One can constrain individual objectives to prevent the search space from growing excessively.

A. NSGA-II and CGP

The NSGA-II algorithm is based on the idea of *Pareto dominance*. The solution p *dominates* the solution q if p is no worse than q in all objectives and p is strictly better than q in at least one objective. The Pareto optimal solutions are not dominated by any other solutions and form the so called *Pareto front*. The individuals in each generation are sorted according to the dominance relation into multiple fronts. The first front F_0 contains all Pareto optimal solutions. Each subsequent front F_i is constructed by removing all the preceding fronts from the population and finding a new Pareto front. Each solution is assigned a *rank* according to the front it belongs to; the solutions from the front F_i have the rank equal to i. The NSGA-II fast non-dominated sort is very efficient, the overall complexity is $\mathcal{O}(MN^2)$, where N is the population size and M is the number of objectives.

The solutions within the individual fronts are sorted according to the *crowding distance* metric, which helps to preserve a reasonable diversity along the fronts [6]. The crowding distance is the average distance of two solutions on either side along each of the objectives. The boundary solutions are assigned an infinite crowding distance, which ensures that these solutions will dominate the inner solutions. Any solution from the front F_i always dominate any solution from F_j, $j > i$. Within the fronts, solutions with higher crowding distance are preferred.

The original NSGA-II algorithm was based on a genetic algorithm, but its extensions for CGP were also introduced [15], [16], [17]. The proposed implementation employs the following modifications. Firstly, due to the absence of the crossover operator in CGP, the offspring population is constructed only using mutation. Secondly, the crowding distance is often not sufficient for CGP to maintain the diversity of the population. As the neutrality present in CGP causes a premature convergence, the Pareto fronts are flooded by individuals that are genotypically distinct but phenotypically identical. Therefore, we introduced a new *equivalence rank*, which enables to put the equivalent solutions in an order and preserve the neutrality character of the CGP [17]. When comparing two individuals, the individual with a lower equivalence rank always dominates the other one. Two individuals with the same equivalence rank are compared using the standard constrained-domination rules. As a consequence, none of the fronts contains individuals with the same fitness and the dominance relation among the individuals with the same fitness is random.

B. Function Set

We used a subset of functions from a generic 180 nm technology process library as the function set in CGP. The function cells have one, two or three inputs (e.g. full adder) and one or two outputs. Complete list of the functions including their area and leakage power can be found in [4]. Some of the functions (e.g. BUF, INV) have multiple sizes which differ in the maximum output load, area, power consumption and delay. During the evaluation, proper size was selected depending on the output load of the gate [4].

C. Output Error

The output error of a candidate circuit is often measured as the number of correct output bits compared to a specified truth table (i.e. the Hamming distance). In the case of approximate circuits, Hamming distance is not often suitable. In this paper, we used the mean relative error:

$$f_{\mathrm{mre}} := \frac{\sum_{\forall i} \frac{\left| O^{(i)}_{\mathrm{orig}} - O^{(i)}_{\mathrm{approx}} \right|}{\max(1, O^{(i)}_{\mathrm{orig}})}}{2^{n_i}}, \tag{1}$$

where $O^{(i)}_{\mathrm{orig}}$ is the decimal representation of the i-th circuit correct output value and $O^{(i)}_{\mathrm{approx}}$ is the individual's i-th output

value. In addition to that, we constrained the worst absolute and relative errors.

D. Circuit Parameters

The area and delay of a candidate circuit were calculated using the parameters defined in the liberty timing file available for the utilized semiconductor technology [4]. The circuit area is estimated as a sum of areas of involved gates. The 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. $t_{\mathrm{d}}(c_i) = f(t_{\mathrm{s}}^{c_i}, C_{\mathrm{l}}^{c_i})$. The delay of the circuit C is determined as the delay of the longest path:

$$\mathrm{Delay}(C) = \max_{\forall p \in \mathrm{path}} \sum_{c_i \in p} t_{\mathrm{d}}(c_i). \tag{2}$$

A detailed description of the power consumption estimation can be found in [4].

E. Initial Population

We used a set of conventional circuits as the initial population. CGP chromosomes for 13 different adder and 6 different multiplier architectures were generated. The adders include Ripple-Carry Adder (RCA), Carry-Select Adder (CSA), Carry-Lookahead Adder (CLA), multiple Tree Adder (TA) and Higher Valency Tree Adder (HVTA) architectures. The multipliers include Ripple-Carry Array, multiple Carry-Save Array and Wallace Tree architectures. The power, area and delay estimates of those circuits can be found in [4].

F. Evolving the Components

Components of two types were designed using the proposed method – approximate 8-bit adders and multipliers. The CGP parameters were set as follows: 500 individuals in the population, 100,000 generations, 10 islands, mutation rate 5 %, the number of rows $n_{\mathrm{r}} = 1$ (to maximize the number of possible connections between blocks). The number of columns was $n_{\mathrm{c}} = 200$ in the case of the adders and $n_{\mathrm{c}} = 1000$ in the case of the multipliers.

The circuits were designed with respect to three objectives – the mean relative error (MRE), power consumption and delay. The MRE was constrained to be at most 10 %, the worst case error was constrained to be at most 5 % of the output range and the worst case relative error was limited to 1000 %, i.e. all candidate solutions violating these requirements were discarded.

Figure 1 shows 473 Pareto optimal 8-bit approximate adders evolved from the initial population of 13 conventional adders. Figure 2 shows 500 Pareto optimal 8-bit approximate multipliers that were evolved from 6 conventional circuits. All parameters are related to the Ripple-Carry Adder and Ripple-Carry Array Multiplier architectures (considered as 100% in the figures), since they are the most power efficient conventional architectures.

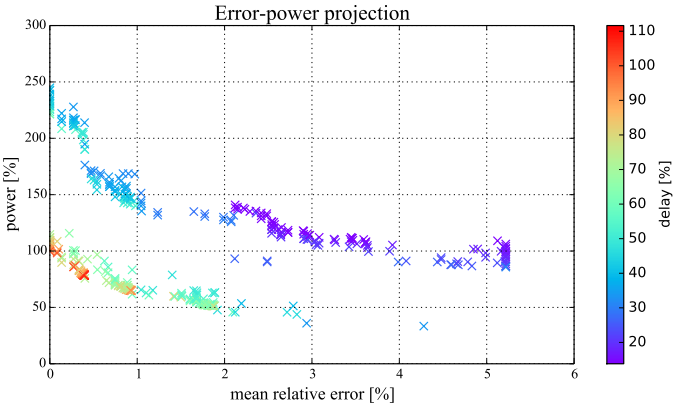


Fig. 1: Pareto front of evolved approximate 8-bit adders.

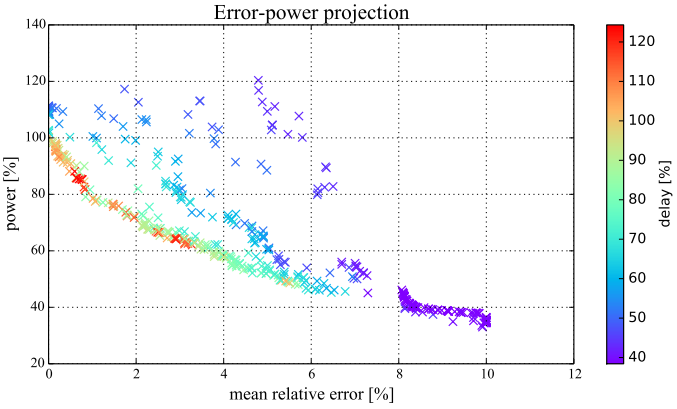


Fig. 2: Pareto front of evolved approximate 8-bit multipliers.

IV. Approximate High Level Synthesis: Binding

The library of evolved approximate components, DFG and maximum latency are the inputs to the proposed method. The objective of the binding process is to find the best assignment of components (functional units) to nodes of the DFG. As our assignment quality metrics consists of three objective functions (area, delay and error), the problem does not necessary have a single overall best solution. Instead the whole set of Pareto optimal solutions have to be considered as the solution of the problem. This way the user can decide which of the objectives is the most important and choose a final solution appropriately.

The binding problem for approximate circuits in the context of HLS is often handled as a knapsack-based optimization problem, where the objective is to maximize energy savings while maintaining a minimal given precision (this leads to the well known 0-1 knapsack problem). In the case of circuits with multiple outputs, this approach leads to Multiple-choice Multiple-dimension Knapsack Problem (MMKP [18]). In this paper, the problem is tackled as a multi-class multi-objective assignment optimization problem, which can be formulated as:

$$\begin{aligned} &min(f_{area}(x), f_{delay}(x), f_{error}(x)) \\ &\text{s.t. } x = (c_{i1}, c_{i2}, \ldots, c_{in}), c_{ij} \in I_j, \end{aligned} \tag{3}$$

where $f_{\_}$ are objective functions, x is a candidate vector

assigning components to DFG nodes and I_j is a set of possible implementations of the component at j-th node.

This formulation of the problem allows us to employ a wide range of optimization algorithms to find the best component assignments. We decided to use the state of the art NSGA-II algorithm which is able to directly optimize against multiple objective functions. Experiments with other multi-objective optimization algorithms are left for future research.

A. Problem Encoding and Genetic Operators

The component assignment problem is encoded with an integer vector of length n, where n is the number of nodes in the DFG. Each item c_i of the vector $x = (c_1, c_2, \ldots, c_n)$, $c_i \in I_i$ assigns a single component to corresponding DFG node. The implementations of components in the library are sorted by their types (adders, multipliers, ...) into sets I_i. This way only components of appropriate type can be assigned to the node.

Our implementation of NSGA-II uses two genetic operators: mutation and crossover. The mutation operator is implemented as a simple random change of the assignment with very low probability per each node i, where another component is selected from corresponding set I_i at random (with the uniform distribution). A simple single point crossover operator is employed. Let $x_A = (c_1^A, c_2^A, \ldots, a_n^A)$ and $x_B = (c_1^B, c_2^B, \ldots, a_n^B)$ be parent assignments, then offspring $x_O = (c_1^A, \ldots, c_p^A, c_{p+1}^B, \ldots, c_n^B)$ is basically combination of x_A, x_B. The crossover point p is randomly selected – again with the uniform distribution.

B. Fitness Function

There are three fitness functions (see def. 3), where the most complicated one is the error metrics $f_{error}(x)$ which will be described separately in Section IV-C.

The total delay of the circuit is computed by adding delays of each step (recall that the input is scheduled DFG), i.e. delays of the slowest components. The overall delay is then defined as

$$f_{delay}(x) = \sum_{s \in S} \max_{\omega \in N_s} \text{Delay}(x(\omega)), \tag{4}$$

where s goes through all scheduled steps, ω goes through all nodes in step s and $\text{Delay}(x(\omega))$ is a delay of the component assigned to node ω.

The total area of the circuit is the sum of all assigned components (note that component sharing is not taken into account):

$$f_{area}(x) = \sum_{\omega \in N} \text{Area}(x(\omega)), \tag{5}$$

where N is a set of all nodes in DFG and $\text{Area}(x(\omega))$ is an area of the component assigned to node ω.

C. Error Metrics

As the input space of complex circuits with multiple inputs is typically vast, the error evaluation is the hardest part of computing the individual's fitness. A typical circuit considered in this study has 8 inputs (usually 8 bits per input, i.e. 2^{8^2} input states in total) which makes the classic error evaluation intractable. This problem is typically tackled by sampling or modeling. Sampling of the input state space is often used in evolutionary design of domain specific circuits, such as image filters, where we can take advantage of knowledge of the input data properties.

Both modeling and sampling are used in this work. The proposed *sampling method* randomly takes k vectors from the input state space and performs simulation of the exact and approximate circuits, followed by comparing their outputs. The input vectors are selected randomly with uniform distribution because no knowledge of the input state space is assumed and incorporated. Figure 3 shows the mean relative standard deviation of the error for a given number of test vectors in the case of DCT-8 benchmark. In the following experiments, we utilized 4,096 test vectors as they provide a good compromise between the error and computational cost.

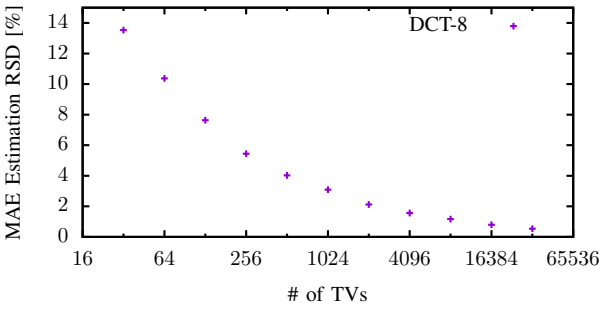


Fig. 3: The mean relative standard deviation of the error for a given number of test vectors

The proposed *modeling approach* is based on a variance error model introduced in [2]. It is able to capture the error propagation in the DFG including structural correlations. The method is based on representation of variables in the computation as random variables (i.e. x becomes $x + \epsilon_x$, where ϵ_x is error). The arithmetic operations are modeled in a similar way so that the error is propagated through the DFG (equations 6 and 7). Authors of [2] assume that only variance of an error is significant (as its constant part can be canceled out with bias) and the output error of the system is then linear combination of random variables. The results of the considered arithmetic operations $(+, \times)$ are expressed as:

$$+ : y + \epsilon_y = (a + \epsilon_a) + (b + \epsilon_b) + \epsilon_+ \tag{6}$$

$$\times : y + \epsilon_y = ab + a\epsilon_b + b\epsilon_a + \epsilon_\times + \cancel{\epsilon_a \epsilon_b} \tag{7}$$

An error sensitivity model which can capture first order structure correlations is used according to [2]. The error variance of an output is defined as:

$$\nu(\epsilon_o) = \sum_{\forall \omega \in V} ES^2_{\omega,o} \cdot \nu(\epsilon_\omega), \tag{8}$$

where ϵ_o is the error at output o and V is a set of all DFG nodes with path to output o. The error sensitivity (ES) is defined for each node as $ES_{\omega,o} = \epsilon_{\omega,o}/\epsilon_\omega$ and can be precomputed in the pre-processing phase. The error sensitivity is computed by a depth first search over DFG going through all paths from output o to node ω. Our error objective function is then defined as the maximal value of $\nu(.)$ across a set of all outputs O of the circuit:

$$f_{error}(x) = \max_{o \in O} \nu(\epsilon_o). \quad (9)$$

D. Experimental Setup

The proposed NSGA-II-based binding algorithm is evaluated using two benchmark problems: a simple 8-to-1 reduction (summing 8 values) and a fixed-point 8-input DCT [19]. Since both circuits operate over 8-bit inputs, they can be built using our library of evolved 8-bit components. Components other than adders and multipliers (such as negations and bit manipulations) are accurate and do not contribute to the error. However, these components represent only a fraction of all components needed to implement DCT.

The experiments begin with the complete library of evolved components containing 473 approximate adders and 500 approximate multipliers (which are all Pareto-optimal). In order to reduce the library size, two methods are proposed and compared: (1) random sampling and (2) taking the best k components according to a scalar fitness:

$$f_x = \sum_{f \in F} w_f \frac{f(x)}{max_{y \in S} f(y) - min_{y \in S} f(y)}, \quad (10)$$

where $F = \{f_{area}, f_{delay}, f_{error}\}$ is a set of objective functions, w_f is weight of normalized objective function $f \in F$, S is a set of all obtained solutions to the binding problem and $f(x)$ is the value of objective function f for solution x. The f_x value is, therefore, the weighted sum of normalized values of objective functions for solution x.

The NSGA-II algorithm operates with a randomly initialized population containing 160 individuals. The number of generations is 5,000 and the probability of mutation is 1%. This setting was fixed after several initial experiments and was leading to a good performance and a reasonable execution time.

E. Results

It is assumed that utilizing all available components provides the best results. The objective is to find a minimal subset of components so that our binding algorithm is able to achieve a sufficient coverage of the solution space. In order to compare the solution space coverage produced by various subsets of components, a "density" indicator is proposed (eq. 11):

$$v_f = \frac{1}{|S| - 1} \sum_{x \in S} min_{y \in S} |F_x - F_y|, x \neq y, \quad (11)$$

where v_f is an average distance between a given solution and its nearest neighbor in the objective function space, $F_x = (f_{area}(x), f_{delay}(x), f_{error}(x))$ is a vector of objective function values for solution x, S is a set of all obtained solutions to the binding problem and $|S|$ is the number of solutions in this set. We should note that the $|S|$ is constant over all of our test cases. Higher values of v_f indicate that solutions are not close to each other and the whole Pareto front can thus be better covered.

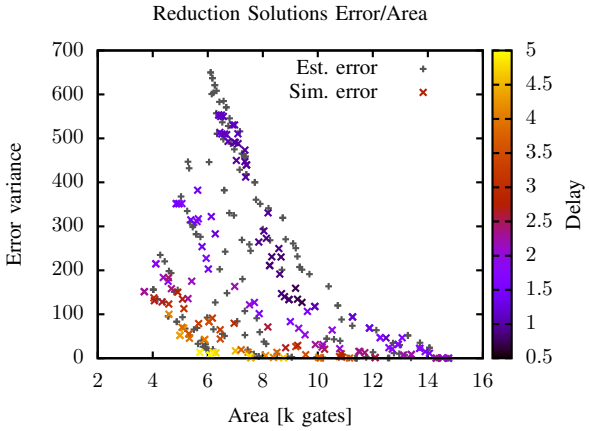


Fig. 4: Pareto fronts obtained using simulation (Sim. error) and statistical model (Est. error) for the Reduce benchmark

1) Experiments with complete library: Figures 4 and 5 show Pareto-optimal sets obtained using the proposed component binding algorithm for the reduce (sum) and DCT-8 flow graphs using the complete component library. In order to compare results, the error is expressed as the error variance for both the sampling method (4,096 test vectors generated) as well as the analytical error model. Note that the output data range is $(0 \ldots 255)$ for Reduction and $(-2 \ldots 2)$ for DCT-8. It can be seen that the analytical error model quite well matches sampled results for the reduction problem, yet shows unacceptable error for the DCT-8. One of the reasons of the model's insufficiency seems to be a fact that the approximate multipliers are used as scale operators ($a * C$, where C is constant), whereas their error was evaluated in the binary configuration ($a * b$). Secondly, the DFG of DCT-8 is much more complex than that of the reduction.

2) Randomly sub-sampled library: In order to find out what is the reasonable (minimal) size of the library that we need in order to achieve similar solution space coverage, we repeated the evolutionary binding task with library sizes from 10 to 640 components (the ratio of adders to multipliers is 1:1). The error is calculated using the statistical model interleaved by simulation.

Results are demonstrated mainly for the DCT-8 benchmark; the Reduce benchmark shows similar characteristics. Figure 6 gives an example – the resulting coverage obtained from three test runs of NSGA-II while using three different samples of 10 components from the complete library. It is apparent that the solution space coverage is rather poor (considering a single run with a single library).

Figure 7 shows how coverage improves for both benchmarks

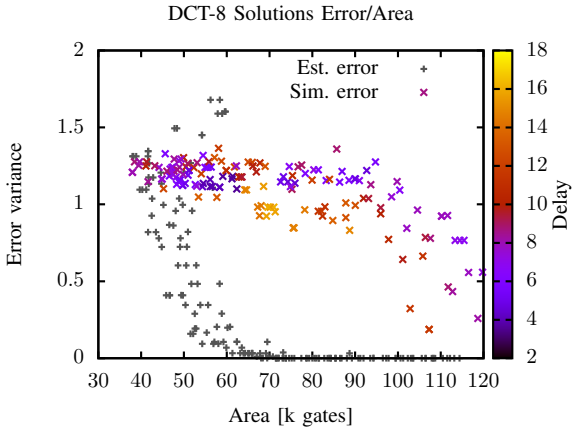


Fig. 5: Pareto fronts obtained using simulation (Sim. error) and statistical model (Est. error) for the DCT-8 benchmark

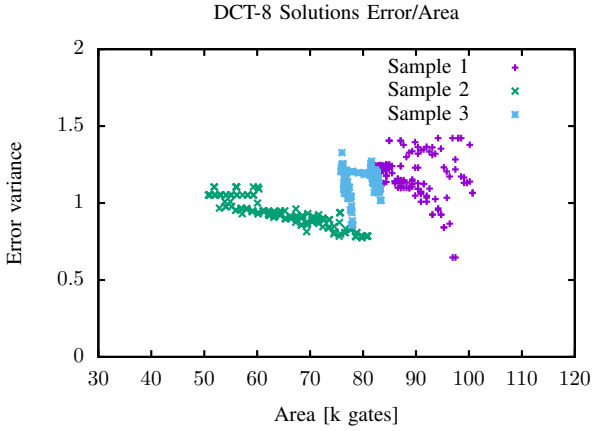
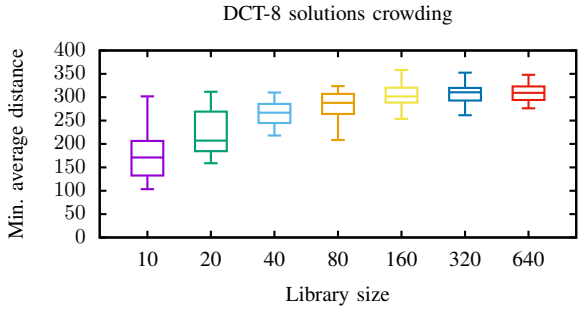


Fig. 6: Random sub-sampling using three different samples of 10 components from the complete library

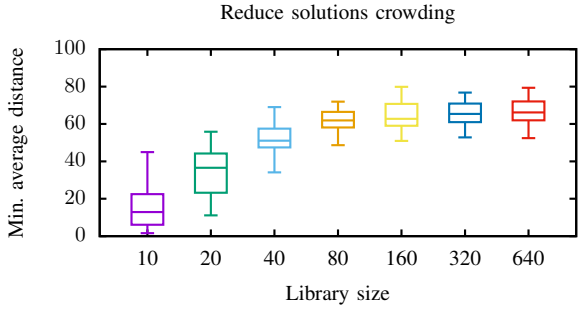
when the library size is increasing. The gains are the most apparent for small library sizes, where only about 80 component implementations are needed to obtain almost the same value of the v_f indicator as for hundreds of components. In these box plots, we used 30 samples per library size (10 different libraries with 3 evolutionary runs per each).

3) Sub-sampled library: Top k components: If we choose k-best components using the scalar fitness (see eq. 10 with $w_1 = w_2 = w_3$), the resulting coverage is slightly worse. Figure 8 and 9 show Pareto-optimal solutions obtained using three smaller library instances and the impact of library size on the v_f indicator. It can be seen that larger component libraries are needed to reach the same v_f values in comparison with the random sub-sampling.

4) Parameters of approximate solutions: Finally, parameters of various evolved implementations of approximate and accurate DCT-8 circuits are summarized in Figure 10. The error is expressed as the mean relative error, which is a standard measure used in approximate computing.



(a)



(b)

Fig. 7: Solution space coverage measured using v_f indicator for different library sizes – Random sub-sampling method

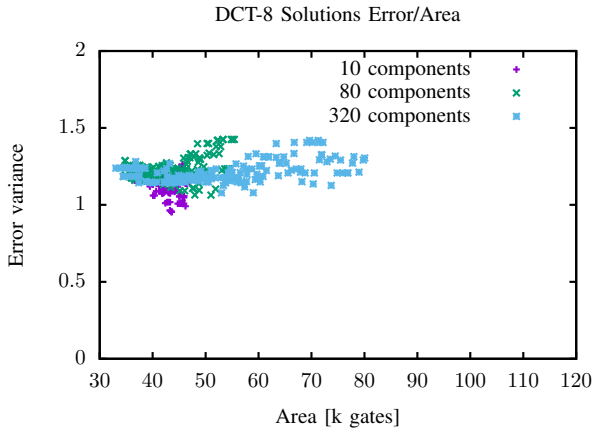


Fig. 8: The Top k components based subsampling using three different samples of 10 components from the complete library

V. Conclusions

In this paper, we employed a multi-objective Cartesian genetic programming to create a comprehensive component library containing hundreds of Pareto optimal implementations of approximate 8-bit adders and multipliers. This component library was utilized in the binding task of approximate HLS, where the binding was conducted using the NSGA-II algorithm. Circuit error was established by means of circuit simulation (based on randomly generated test vectors) and

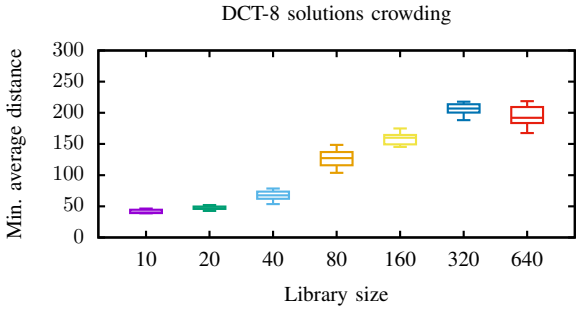


Fig. 9: Solution space coverage measured using v_f indicator for different library sizes – Top k components method

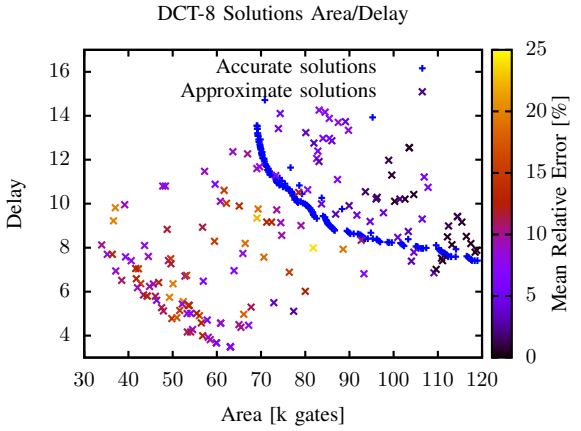


Fig. 10: Parameters of various evolved implementations of approximate and accurate DCT-8 circuits.

statistical modeling. Combining the circuit simulation with statistical modeling is the preferred method as the statistical error estimation shows some drawbacks for more complex circuits.

Two approaches were evaluated in order to reduce the size of the library of approximate components. It turns out under conditions of our experiments that the random sub-sampling of the component library allows for better coverage of the solution space for both benchmarks. The Top k selection seems to be too biased by the approach we used to scalarize the objectives.

This initial study left open many issues. Our future work will be devoted to improving the statistical error estimation (dealing with error estimation for specific components), evaluating more complex benchmark circuits, analyzing the impact of NSGA-II parameters on the quality of results and proposing other component library reducing methods based on gained experience.

Acknowledgment

This work was supported by the Czech science foundation project GA16-17538S and by The Ministry of Education, Youth and Sports of the Czech Republic from the National Programme of Sustainability (NPU II); project IT4Innovations excellence in science - LQ1602.

References

[1] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, p. 62:162:33, 2016.

[2] C. Li, W. Luo, S. S. Sapatnekar, and J. Hu, "Joint precision optimization and high level synthesis for approximate computing," in *Proc. of the 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. ACM, 2015, pp. 1–6.

[3] Z. Vasicek and L. Sekanina, "Evolutionary approach to approximate digital circuits design," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 432–444, 2015.

[4] R. Hrbacek, V. Mrazek, and Z. Vasicek, "Automatic design of approximate circuits by means ofmulti-objective evolutionary algorithms," in *Proceedings of the 11th Int. Conf. on Design and Technology of Integrated Systems in Nanoscale Era*. IEEE, 2016, pp. 239–244.

[5] Z. Vasicek and L. Sekanina, "Evolutionary design of complex approximate combinational circuits," *Genetic Programming and Evolvable Machines*, vol. 17, no. 2, pp. 169–192, 2016.

[6] 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.

[7] R. Nane, V. M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels, "A survey and evaluation of fpga high-level synthesis tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, 2016.

[8] P. Coussy, D. D. Gajski, M. Meredith, and A. Takach, "An introduction to high-level synthesis," *IEEE Design Test of Computers*, vol. 26, no. 4, pp. 8–17, 2009.

[9] 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.

[10] 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.

[11] 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.

[12] 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, 2013, pp. 1367–1372.

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

[14] M. Soeken, D. Grosse, A. Chandrasekharan, and R. Drechsler, "BDD minimization for approximate computing," in *21st Asia and South Pacific Design Automation Conference ASP-DAC 2016*. IEEE, 2016, pp. 474–479.

[15] P. Kaufmann, T. Knieper, and M. Platzner, "A novel hybrid evolutionary strategy and its periodization with multi-objective genetic optimizers," in *IEEE Congress on Evolutionary Computation*. IEEE, 2010, pp. 1–8.

[16] J. A. Hilder and J. A. W. andAndy M. Tyrrell, "Use of a multi-objective fitness function to improve cartesian genetic programming circuits," in *2010 NASA/ESA Conference on Adaptive Hardware and Systems*. IEEE, 2010, pp. 179–185.

[17] R. Hrbacek, "Parallel multi-objective evolutionary design of approximate circuits," in *GECCO'15 Proceedings of the 2015 conference on Genetic and evolutionary computation*. ACM, 2015, pp. 687–694.

[18] M. R. Razzazi and T. Ghasemi, "An exact algorithm for the multiple-choice multidimensional knapsack based on the core," in *Advances in Computer Science and Engineering*. Springer, 2008, pp. 275–282.

[19] A. Yukihiro, A. Takeshi, and M. Nakajima, "A fast DCT-SQ scheme for images," *IEICE TRANSACTIONS (1976-1990)*, vol. 71, no. 11, pp. 1095–1097, 1988.