
**BRNO UNIVERSITY OF TECHNOLOGY**
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

**FACULTY OF INFORMATION TECHNOLOGY**
**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**

**DEPARTMENT OF COMPUTER SYSTEMS**
**ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ**


# AUTOMATED MULTI-OBJECTIVE PARALLEL EVOLUTIONARY CIRCUIT DESIGN AND APPROXIMATION

**AUTOMATICKÝ MULTIKRITERIÁLNÍ PARALELNÍ EVOLUČNÍ NÁVRH A APROXIMACE OBVODŮ**

**PHD THESIS**
**DIZERTAČNÍ PRÁCE**


**AUTHOR** — **Ing. RADEK HRBÁČEK**
**AUTOR PRÁCE**

**SUPERVISOR** — **Prof. Ing. LUKÁŠ SEKANINA, Ph.D.**
**VEDOUCÍ PRÁCE**



**BRNO 2017**

# Abstract

Recently, energy efficiency has become one of the most important properties of computing platforms, especially because of limited power supply capacity of battery-power devices and very high consumption of growing data centers and cloud infrastructure. At the same time, in an increasing number of applications users are able to tolerate inaccurate or incorrect computations to a certain extent due to the imperfections of human senses, statistical nature of data processing, noisy input data etc.

Approximate computing, an emerging paradigm in computer engineering, takes advantage of relaxed functionality requirements to make computer systems more efficient in terms of energy consumption, computing performance or complexity. Error resilient applications can achieve significant savings while still serving their purpose with the same or a slightly degraded quality.

Even though new design methods for approximate computing are emerging, there is a lack of methods for automated approximate HW/SW design offering a rich set of compromise solutions. Conventional methods often produce solutions that are far from an optimum. Evolutionary algorithms have been shown to bring innovative solutions to complex design and optimization problems. However, these methods suffer from several problems, such as the scalability or a high number of fitness evaluations needed to evolve competitive results. Finally, existing methods are usually single-objective whilst multi-objective approach is more suitable in the case of approximate computing.

In this thesis, a new automated multi-objective parallel evolutionary algorithm for circuit design and approximation is proposed. The method is based on Cartesian Genetic Programming. In order to improve the scalability of the algorithm, a brand new highly parallel implementation was proposed. The principles of the NSGA-II algorithm were used to provide the multi-objective design and approximation capability.

The performance of the implementation was evaluated in multiple different applications, in particular (approximate) combinational arithmetic circuits design, bent Boolean functions discovery and approximate logic circuits for TMR schema. In these cases, important improvements with respect to the state of the art were obtained.



# Keywords

Approximate Computing, Approximate Circuits, Digital Circuits, Evolutionary Algorithms, Evolutionary Design, Cartesian Genetic Programming, Multi-Objective Optimization


# Citation

Radek Hrbáček. *Automated Multi-Objective Parallel Evolutionary Circuit Design and Approximation*, PhD thesis, Department of Computer Systems, Faculty of Information Technology, Brno University of Technology, Brno, CZ, 2017

# Abstrakt

Spotřeba a energetická efektivita se stává jedním z nejdůležitějších parametrů při návrhu počítačových systémů, zejména kvůli omezené kapacitě napájení u zařízení napájených bateriemi a velmi vysoké spotřebě energie rostoucích datacenter a cloudové infrastruktury. Současně jsou uživatelé ochotni do určité míry tolerovat nepřesné nebo chybné výpočty v roustoucím počtu aplikací díky nedokonalostem lidských smyslů, statistické povaze výpočtů, šumu ve vstupních datech apod.

Přibližné počítání, nová oblast výzkumu v počítačovém inženýrství, využívá rozvolnění požadavků na funkčnost za účelem zvýšení efektivity počítačových systémů, pokud jde o spotřebu energie, výpočetní výkon či složitost. Aplikace tolerující chyby mohou být implementovány efektivněji a stále sloužit svému účelu se stejnou nebo mírně sníženou kvalitou.

Ačkoli se objevují nové metody pro návrh přibližně počítajících výpočetních systémů, je stále nedostatek automatických návrhových metod, které by nabízely velké množství kompromisních řešení dané úlohy. Konvenční metody navíc často produkují řešení, která jsou daleko od optima. Evoluční algoritmy sice přinášejí inovativní řešení složitých optimalizačních a návrhových problémů, nicméně trpí několika nedostatky, např. nízkou škálovatelností či vysokým počtem generací nutných k dosažení konkurenceschopných výsledků. Pro přibližné počítání je vhodný zejména multikriteriální návrh, což existující metody většinou nepodporují.

V této práci je představen nový automatický multikriteriální paralelní evoluční algoritmus pro návrh a aproximaci digitálních obvodů. Metoda je založena na kartézském genetickém programování, pro zvýšení škálovatelnosti byla navržena nová vysoce paralelizovaná implementace. Multikriteriální návrh byl založen na principech algoritmu NSGA-II.

Výkonnost implementace byla vyhodnocena na několika různých úlohách, konkrétně při návrhu (přibližně počítajících) aritmetických obvodů, Booleovských funkcích s vysokou nelinearitou či přibližných logických obvodů pro tří-modulovou redundanci. V těchto úlohách bylo dosaženo význammých zlepšení ve srovnání se současnými metodami.



# Klíčová slova

přibližné počítání, přibližné obvody, logické obvody, evoluční algoritmus, evoluční návrh, kartézské genetické programování, multikriteriální optimalizace


# Citace

Radek Hrbáček. *Automatický multikriteriální paralelní evoluční návrh a aproximace obvodů*, Dizertační práce, Ústav počítačových systémů, Fakulta informačních technologií, Vysoké učení technické v Brně, Brno, CZ, 2017

# Automated Multi-Objective Parallel Evolutionary Circuit Design and Approximation

## Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně pod vedením Prof. Ing. Lukáše Sekaniny, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Radek Hrbáček
15. června 2017

# Contents

# Chapter 1

# Introduction

This chapter gives an introduction to the thesis. It starts with the motivation for the whole research, then the open problems and research objectives of the thesis are formulated. At the end of the chapter, an outline of the thesis is given.

## 1.1 Motivation

Computers and computer based systems play a crucial role in people's everyday lives. Embedded systems can be found almost everywhere. Power efficiency is becoming increasingly important property of computing platforms, especially because of limited power supply capacity of embedded devices and high costs associated with operating growing data centers and cloud infrastructure. At the same time, in an increasing number of applications users are able to tolerate inaccurate or incorrect computations to a certain extent due to imperfections of human senses, statistical nature of data processing, noisy input data etc.

*Approximate computing*, an emerging paradigm in computer engineering, takes advantage of relaxed functionality requirements to make computer systems more efficient in terms of energy consumption, computing speed or complexity [42]. Error resilient applications can achieve significant savings while still serving their purpose with the same or a slightly degraded quality.

The complexity of computer systems is permanently growing and thus, automated design tools have to deal with more complex problems specified on higher level of abstraction than before. The same holds true for approximate computing. Even though new methods are emerging, there is a lack of methods for automated approximate HW/SW design offering a rich set of compromise solutions. Moreover, conventional synthesis algorithms often produce solutions that are far from an optimum [10].

*Evolutionary algorithms* (EAs) have been confirmed to bring innovative solutions to complex design and optimization problems. Recently, complex digital circuits have been optimized by means of EAs while the scalability of the method has been improved substantially [27, 69].

Every year, a special competition *Humies* is held at the Genetic and Evolutionary Computation Conference to award scientific results that utilize an evolutionary computation technique and are *human-competitive* [35]. In years 2004-2013, there were 42 Humie winners and 10 of them published results that were patented or would qualify as a patentable new invention [31]. The same trend can be observed for years 2014-2016.

## 1.2 Open Problems

The main issue of approximate computing at the moment is the lack of available automated methods capable of providing approximations for arbitrary combinational circuits under different error metrics and with respect to multiple objectives. The solutions provided by conventional circuit synthesis methods are often far from optimum.

On the other hand, the evolutionary based design and approximation methods suffer from several problems, mainly the scalability of the methods (i.e. the scalability of the fitness function and the representation of candidate circuits) is not sufficient. A high number of fitness evaluations needed to evolve competitive results implies simplifications in circuit parameters estimation and thus leads to reduced accuracy of the estimations. Although complex digital circuits have been optimized using single-objective EAs, the same cannot be said about the multi-objective methods.

## 1.3 Research Objectives

The first main research objective for this thesis is to

> *develop an automated scalable design method based on evolutionary algorithms, capable of multi-objective design and approximation of digital circuits.*

As indicated, such a method has to meet several requirements. It has to be able to design circuits of a sufficient complexity. It has to take into account multiple design criteria. Moreover, the estimation of the circuit parameters has to be accurate enough. Finally, the implementation should be parallelized and should efficiently utilize computational resources.

The second main objective is to

> *show on several real-world problems that the method provides human-competitive results.*

These objectives can be translated into the following partial goals:

1. To develop an optimized parallel evolutionary algorithm for digital circuits design.
2. To extend the evolutionary design method with multi-objective design capability.
3. To identify objectives relevant for approximate circuits and transform them to fitness functions.
4. To carry out experiments on different real world applications to show the performance of the method.
5. To validate the achieved results by means of professional simulation tools.

## 1.4 Thesis Outline

The thesis is composed as a collection of papers. The research contribution of this thesis is comprised of seven peer-reviewed research papers in their original publication format attached in Appendix A. The thesis is organized as follows: Chapter 1 gives an introduction to the thesis. Chapter 2 surveys the state of the art and presents relevant background

information for the research. Chapter 3 summarizes the research process and gives an overview over the papers constituting the research contribution. Finally, Chapter 4 presents conclusions and proposes future research directions.

# Chapter 2

# Survey of the State of the Art

This chapter gives relevant background information needed for a proper understanding of the work presented in the thesis. It primarily addresses the areas of approximate computing and evolutionary circuit design.

## 2.1 Approximate Computing

Recently, power efficiency has become one of the most important parameters of almost every computing platform. At the same time, a wide range of applications in which we are willing to tolerate imperfections in computations has spread out. As a consequence, a new research field – *approximate computing* – has emerged to investigate how computer system can be made more efficient in terms of energy consumption, computing speed or complexity assuming that some errors are acceptable. It has been believed, that significant savings can be achieved by relaxing the requirement of perfect functionality thanks to the *error resilience* of some applications. Therefore, the *accuracy* (error) of the system can be used as a design metric and inaccurate solutions can be accepted if an improvement in other parameters occurs.

The approximation can be introduced at various levels including the entire computer system architecture [38], particular components (e.g. ALU) [15, 37], operating system, algorithm or even programming language [54, 3]. As the complexity of today's computer systems grows, manual approximation is not an efficient design method. Hence, several automated approximate design methods have been introduced.

### 2.1.1 Application Error Resilience

Inherent application *error resilience* is the property of an application to produce acceptable outputs even if some underlying computations are approximate or incorrect [6]. Whether an output is acceptable or not is given by an *output quality* metric if the concept of approximate computing is considered. Applications are designed to produce outputs of acceptable quality rather than a unique correct output.

The sources that contribute to the application resilience can be classified into following the categories [6]:

- *Inputs*: Applications that process noisy or redundant data can be inherently resilient.

- *Outputs*: If the specification does not define a unique golden output or the outputs are consumed by human senses, minor output variations that are often indistinguishable are acceptable.

- *Computation patterns*: Statistical computations can result in attenuation or cancellation of error. Applications employing statistical computations are thus resilient.

- *Iterative processing*: Many applications feature iterative processing and undergo successive refinement or aggregation to obtain converged results. The quality gain tends to attenuate as the iterations continue [82].

The level, at which the system is approximated, influences the resilience to approximations. For example, introducing approximations at the software level can lead to a very different conclusion about the error resilience than at the hardware level [6].

The overall output quality is given by individual responses to different system inputs, which makes the output quality a statistic. In general, the most frequently used quality metrics are the *error probability* (error rate), *error magnitude* (mean error) and *error predictability* (error variance) [6]. These metrics form a three-dimensional space and all acceptable qualities form a subspace in this space. This subspace is highly application dependent, but in general, a wide range of applications accept outputs with low error rate or low error magnitude [6].

The computation patterns (e.g. in statistical processing) present in particular applications affect the error resilience significantly. Thanks to that, there are applications that accept output errors with small variance present in all computations (which correspond to very high error rate) while the error magnitude can be very large. In addition to computation patterns, the context in which the application is used significantly impacts the resilience. For example, the error resilience of a k-means clustering algorithm used in an image segmentation application depends on chosen quality metric. The application is able to tolerate more aggressive approximations if mean centroid distance is used as the quality metric in comparison with the percentage of mis-clustered points [6].

Approximations can be done at multiple levels by applying several approximate computing techniques at the same time. The resilience is again application dependent in such situation. Generally, different approximation techniques can be applied in a synergistic manner, but there can be cases for which the combination of particular techniques leads to unacceptable results [6].

### 2.1.2 Approximate Circuits

While automatic design of digital circuits has been well established in the past, the correct functionality has always been an essential requirement put on the circuits [7]. The other parameters, like the area, delay or power consumption, have been considered as secondary and have not been optimized as long as a fully working solution has been found.

The *approximate circuits* are designed in such a way that the functionality specification (assuming a perfect operation) is not fully met in exchange for savings in terms of area, delay, power consumption etc. Although the circuit is not working properly, it can still be suitable for applications in which certain level of error is not recognizable (e.g. human perception in the context of multimedia applications). Moreover, in some cases (e.g. low battery), users could knowingly tolerate even more inaccuracy in order to extend the time of operation.

### 2.1.3 Design Objectives

When designing an approximate computer system, the functionality requirement (*accuracy*) is traded off for improvements in other design objectives. These objectives are application dependent, but they usually include *size*, *power consumption* and *performance*. In the case of hardware solutions, one has to deal with the *reliability and dependability* of the system. In order to correctly determine parameters (i.e. particular values for the objectives) of computer systems, careful benchmarking has to be performed. The acceptable level of inaccuracy is application dependent.

#### Accuracy/Error

The accuracy (error) of a computer system is the main objective tracked when doing approximations. In each application, different requirements on the accuracy metric can be formulated resulting in a wide range of different accuracy metrics being used. Usually, an opposite metric, the *error*, is used instead of accuracy.

For combinational circuits, one can use the number of incorrect outputs (i.e. the *Hamming distance*):

$$e_{\text{hamm}} = \sum_{\forall i, \forall j, O_{\text{approx}}^{(i,j)} \neq O_{\text{orig}}^{(i,j)}} 1, \tag{2.1}$$

where $O_{\text{approx}}^{(i,j)}$ is the $j$-th bit of the $i$-th circuit output and $O_{\text{orig}}^{(i,j)}$ is the correct one. The other option is to calculate the error probability [68]:

$$e_{\text{prob}} = \frac{\sum_{\forall i, O_{\text{approx}}^{(i)} \neq O_{\text{orig}}^{(i)}} 1}{2^{n_{\text{i}}}}, \tag{2.2}$$

where $n_{\text{i}}$ is the number of circuit inputs, $O_{\text{approx}}^{(i)}$ is the $i$-th circuit output (all bits) and $O_{\text{orig}}^{(i)}$ is the correct one.

The significance of the aforementioned two metrics is very low for arithmetic circuits or digital signal processing systems in general, for which more suitable metrics based on the arithmetic distance between the actual and correct values exist. One can use the worst case error $e_{\text{wce}}$, the mean absolute error $e_{\text{mae}}$, the mean squared error $e_{\text{mse}}$ or their relative versions $e_{\text{wcre}}$ and $e_{\text{mre}}$ as follows [68]:

$$e_{\text{wce}} = \max_{\forall i} \left| O_{\text{approx}}^{(i)} - O_{\text{orig}}^{(i)} \right|, \tag{2.3}$$

$$e_{\text{mae}} = \frac{\sum_{\forall i} \left| O_{\text{approx}}^{(i)} - O_{\text{orig}}^{(i)} \right|}{2^{n_{\text{i}}}}, \tag{2.4}$$

$$e_{\text{mse}} = \frac{\sum_{\forall i} \left| O_{\text{approx}}^{(i)} - O_{\text{orig}}^{(i)} \right|^2}{2^{n_{\text{i}}}}, \tag{2.5}$$

$$e_{\text{wcre}} = \max_{\forall i} \frac{\left| O_{\text{approx}}^{(i)} - O_{\text{orig}}^{(i)} \right|}{\max(1, O_{\text{orig}}^{(i)})}, \tag{2.6}$$

$$e_{\text{mre}} = \frac{\sum_{\forall i} \frac{\left| O_{\text{approx}}^{(i)} - O_{\text{orig}}^{(i)} \right|}{\max(1, O_{\text{orig}}^{(i)})}}{2^{n_{\text{i}}}}, \tag{2.7}$$

where $n_{\text{o}}$ is the number of circuit outputs.

### Size

The size corresponds to the amount of resources statically used by the system. Computer programs vary in the amount of program memory occupied or data memory operating on, digital circuits occupy an area on die or the number of used reconfigurable cells on Field Programmable Gate Arrays (FPGAs).

### Power Consumption

The power consumption is the most important objective being shrunk in approximate computing. It can be measured as energy needed for performing an operation or as average power consumption of a permanently running system.

In the case of digital circuits (CMOS technology), there are three major sources of power dissipation [5]:

$$\begin{aligned} P_{\text{avg}} &= P_{\text{switching}} + P_{\text{short\_circuit}} + P_{\text{leakage}} \\ &= \alpha \cdot C_{\text{L}} \cdot V_{\text{dd}}^2 \cdot f_{\text{clk}} + I_{\text{sc}} \cdot V_{\text{dd}} + I_{\text{leakage}} \cdot V_{\text{dd}}. \end{aligned} \tag{2.8}$$

The switching power $P_{\text{switching}}$ depends on the switching activity $\alpha$ (probability of switching a gate's output), the operating frequency $f_{\text{clk}}$, load capacitance $C_{\text{L}}$ and power supply voltage $V_{\text{dd}}$. The second term, the short circuit power $P_{\text{short\_circuit}}$, is caused by the short circuit current $I_{\text{sc}}$ which arises when both complementary transistors are active at the same time, i.e. conducting current directly from supply to ground. These two terms together represent the dynamic power, while the switching power is usually 10 times greater than the short circuit power. The last term is the leakage (static) power $P_{\text{leakage}}$, which is primarily determined by fabrication technology considerations [5].

The introduction of CMOS technology led to significant reductions in static power consumption. For a long time, the dynamic power dissipation was the dominant component. However, with the decreasing size of the semiconductor technology process, the static dissipation is increasing due to rising leakage currents and is becoming the major component of the power consumption [62].

### Performance

The next commonly used objective is the performance (speed) of computation. In the case of computer programs, it usually applies to the execution time. The speed of digital circuits can be determined by the maximum operating frequency or the latency, i.e. the interval between the stimulation of the inputs and the response on the outputs. Another way of measuring the speed is to compute the throughput of the system.

### Reliability/Dependability

Reducing the probability of failure and increasing the reliability of digital circuits is an important design objective. Many applications (e.g. automotive, aerospace, operating in remote environments) are safety-critical and need to be built using the principles of fault-tolerant system design. As the complexity of computer systems increases, more complex mechanisms must be introduced to preserve the reliability of the systems [62].

### 2.1.4 Overview of Circuit Approximation Methods

Automated approximate computing techniques are being developed to speed up the design process and to find the best trade-off solutions between the resources being shrunk and the inaccuracy of the computation. The design of approximate circuits is typically based on modifying fully functional circuits [59]. Most of the methods deal with combinational circuit design, however, there are methods for sequential circuit approximation as well.

#### Voltage Over-Scaling

As can be seen from Equation 2.8, the power consumption $P_{\text{avg}}$ is highly dependent on the supply voltage $V_{\text{dd}}$. Present computer systems often utilize voltage scaling together with frequency scaling in order to lower the power consumption when full performance is not needed. Voltage over-scaling extends this concept beyond the critical voltage value at which the critical path delay is just met. This leads to significant energy savings for the price of possible incorrect computations [32].

The supply voltage can be controlled adaptively with respect to the occurrence of errors in the circuit. For example, the adaptive voltage over-scaling strategy presented in [36] monitors several locations in the circuit where errors can be detected. The signals are sampled with a delayed clock and compared to the value sampled with the main clock. If they differ, an error is detected. The supply voltage is then controlled according to current error rate.

The drawback of the voltage over-scaling approach is the difficulty of controlling the error. Since the behavior of the circuit after voltage over-scaling depends on many factors (each logic gate behaves differently according to its type, input timing, output load, etc.), accurate timing analysis has to be performed so as to measure the output quality. For example, the Modeling and Analysis of Circuits for Approximate Computing (MACACO) methodology is based on the construction of an equivalent circuit that represents the behavior of the approximate circuit at a given voltage and clock frequency [77].

#### Manual Methods

In the first approximation methods, the design of approximate circuits was typically based on manual modifications of fully functional circuits. These first results include arithmetic circuits, such as combinational adders [15, 16] or multipliers [37]. In general, only small components have been approximated manually, e.g. 2-bit multiplier occupying nearly half area and working almost correctly except for a single output value ($3 \cdot 3 = 7$) [37]. By using this simple component as a building block, one can design larger circuits, however, the method clearly does not exploit the whole potential of approximate computing.

#### SALSA

The Systematic methodology for Automatic Logic Synthesis (SALSA) uses a quality function which decides whether a predefined quality constraint is met or not. The algorithm is allowed to modify the circuit as long as the quality constraint is not exceeded. SALSA has been applied to a number of problems, e.g. 32-bit adders, 8-bit multipliers, FIR filters, DCT blocks and others [75].

### SASIMI

Another approach, Substitute-and-Simplify (SASIMI), looks for signal pairs having similar values with a high probability. By substituting one signal for the other, a part of the circuit can be removed resulting in area and power savings at the cost of an error introduced to the circuit. Moreover, SASIMI further extends the approach to synthesize quality configurable circuits, where at runtime, processing of selected input vectors is given an additional cycle to correct errors due to approximations [76].

### ABACUS

Unlike the aforementioned methods, ABACUS (Automated Behavioral Approximate Circuit Synthesis) operates directly on the behavioral descriptions of circuits. ABACUS automatically generates approximate circuits from input behavioral descriptions by performing global transformations on an abstract synthesis tree (AST) created from the behavioral description. The outcome approximate circuits are still expressed in behavioral code and can be synthesized by means of standard synthesis tools. Complementary approximate computing methods, e.g. voltage over-scaling or manually created approximate components, may be still used [45]. The latest version of the algorithm supports multi-objective design based on the principles of the NSGA-II algorithm [46].

### ASLAN

Although most of the design methods deal with combinational circuits, there are methods capable of approximating sequential circuits. As an example, the Automatic Methodology for Sequential Logic Approximation (ASLAN) creates an approximate version of a sequential circuit that consumes lower energy, while meeting a specified quality constraint. ASLAN identifies combinational blocks in the sequential circuit that are amenable to approximation and iteratively approximates the entire sequential circuit using a gradient-descent approach [52].

### EA-based Methods

Several evolutionary algorithm based methods have been used in approximate computing recently. Most of the methods are single-objective and the optimization of a secondary objective is achieved either by restricting the circuit resources (by constraining the genotype size) [58, 67] or using a multi-phase approach [68]. A multi-objective evolutionary algorithm was used to design approximate multiple constant multipliers [48]. However, the method operated on functional unit level and the complexity of the circuits was relatively small.

### Summary

Despite numerous attempts, almost all papers dealing with the design of approximate circuits show some of the following features that are undesirable [44]:

- The approximation method is described, but a corresponding software implementation is not available.
- An implementation of the original (accurate) circuit is not available.

- The quality of approximation and other parameters of approximate circuits are expressed relatively to parameters of the original circuits.
- Implementations of the resulting approximate circuits are not available.
- Only a few approximate versions created from the original circuit are reported, forming thus a sparsely occupied Pareto front.
- It is unclear if a given number of test vectors used to evaluate approximate circuits is sufficient for obtaining a trustworthy error quantification if the error is determined using simulation.
- A given approximation method is only rarely compared against competitive approximation methods.

## 2.2 Evolutionary Design

Evolutionary Algorithms (EAs), generic population-based metaheuristic optimization algorithms, use mechanisms inspired by biological evolution, such as reproduction, recombination, mutation or selection for purposes of optimization and design. Population of individuals represents a set of candidate solutions to a specified problem. Each individual is assigned a *fitness value* depending on the ability of the individual to solve the problem. In each generation, a subset of the population is selected according to the fitness value to create offspring population by means of recombination and mutation [1].

While EAs were originally used to solve optimization problems, they are able to bring innovative solutions to design problems as well. Evolutionary design of hardware is a growing research area since the beginning of the 1990s. In particular, it includes evolutionary design of digital and analog circuits, antennas, optical systems and microelectromechanical systems (MEMS) [56].

EAs have been applied to a number of real problems, however, their computational complexity can be enormous. The scalability of the fitness function is often a prohibiting factor and thus, one has to deal with the acceleration of the fitness function or fitness approximation. Besides the scalability of the fitness evaluation, another problems that limit the application of EAs are known, such as the scalability of the representation (complex problems are represented by long chromosomes which implies large search space), the nondeterministic nature of EAs or slow convergence. Potential solutions to the problems have been recently summarized in [62].

In the following section, we will introduce Cartesian Genetic Programming (CGP) since it has been routinely used in the area of evolutionary based digital circuit design and optimization.

### 2.2.1 Cartesian Genetic Programming

Cartesian genetic programming has been introduced by Miller [41] as a branch of genetic programming. Unlike GP which uses tree representation, an individual in CGP is represented by a directed acyclic graph which enables the candidate solution to have multiple outputs and automatically reuse intermediate results. This makes CGP very suitable for the design of various kinds of digital circuits (such as arithmetic and logic circuits, digital filters, etc.) and computer programs [39].

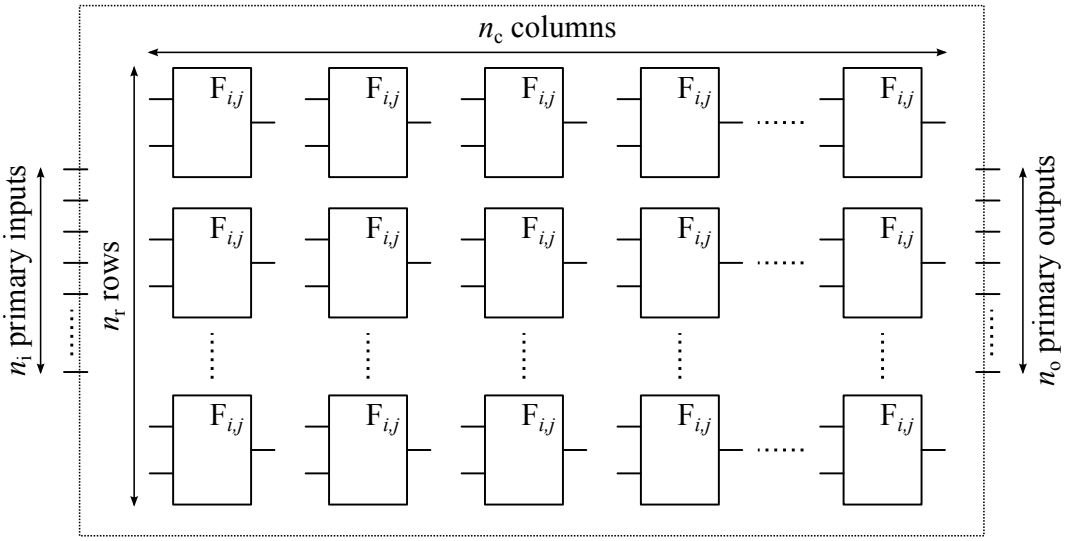



Figure 2.1: Cartesian genetic programming scheme.

CGP uses a cartesian grid of $n_r \times n_c$ programmable elements (nodes) interconnected by a feed-forward network (Figure 2.1). Each node's input (each node has a fixed number of inputs, e.g. $n_{ni} = 2$) can be connected either to one of $n_i$ primary inputs or to a node output in the preceding $l$ columns. By setting the $l$-back parameter and the grid size, one can control the area and delay of the circuit. Each node can be programmed to perform one of $n_{ni}$-input functions defined in the set $\Gamma$ (let $n_f = |\Gamma|$). The $n_o$ primary circuit outputs are connected either to the primary inputs or nodes. The output connectivity can be optionally restricted by the $o$-back parameter.

Since all the CGP parameters are fixed, each chromosome is encoded using a fixed-size string of $n_r \cdot n_c \cdot (n_{ni} + 1) + n_o$ integers. Each primary input is assigned a number from $\{0, ..., n_i - 1\}$ and the nodes are assigned numbers from $\{n_i, ..., n_i + n_r \cdot n_c - 1\}$. The genotype is of fixed length, whereas the phenotype is of variable length depending on the number of inactive nodes, i.e. nodes whose output is not used by any other node or primary output. Hence, the genotype-phenotype mapping is not injective. The existence of genotypes with the same fitness is usually referred to as neutrality. The role of neutrality has been intensively studied [83] and it was shown that for certain problems the neutrality significantly reduces the computational effort and helps to find more innovative solutions [40].

In CGP, a simple mutation based $(1+\lambda)$ evolutionary strategy is used as a search mechanism. The population size $1 + \lambda$ is usually very small, typically, $\lambda$ is between 1 and 15. The initial population is constructed either randomly (then we speak about evolutionary design) or by mapping of a known solution to the CGP chromosome (evolutionary optimization) [69]. In each generation, a randomly selected individual with the best fitness value (if there are more of them, an individual genotypically distinct from the parent) is passed to the next generation unmodified and its $\lambda$ offspring individuals are created by means of point mutation operator which modifies $m$ randomly selected genes of the chromosome. The mutation rate is usually set to modify up to 5 % of the total number of genes. For some problem classes (e.g. symbolic regression problem), special crossover operators have been investigated [8], however, none of them has been confirmed to significantly improve the search process.

Recently, several modifications to CGP have been published. Embedded CGP is an extension of the CGP which is capable of automatically acquiring, evolving and re-using partial solutions in the form of modules [78]. By introducing multiple chromosomes, each connected to a single output, large problems with multiple outputs can be broken down into many smaller problems leading to significant performance increase for particular problems [79, 80]. Self Modifying CGP enables self-modifications of CGP individuals by introducing operations into the CGP chromosome [17, 18, 19, 20]. Multi Expression CGP modifies CGP individual evaluation in such a way that multiple nodes are compared to the desired output instead of just a single node [4]. Recurrent CGP allows recurrent connections [65]. All these modifications share a common objective – to increase the scalability of CGP and speed up the evolutionary process.

### 2.2.2 Evolutionary Design of Digital Circuits

In the case of combinational circuit evolution, the fitness function corresponds to the quality of the candidate circuit measured as the number of correct output bits compared to a specified truth table (see Equation 2.1). In order to obtain a fully working circuit, all combinations of input values have to be evaluated. For a circuit with $n_\mathrm{i}$ inputs and $n_\mathrm{o}$ outputs, $2^{n_\mathrm{i}}$ test vectors need to be fetched to the primary inputs and $n_\mathrm{o} \cdot 2^{n_\mathrm{i}}$ output bits have to be verified so as to compute the fitness value. In this thesis, we assume this scenario.

Recently, complex digital circuits have been successfully optimized by means of CGP [72]. However, designing complex circuits from scratch (from a randomly generated initial population) has been shown to be much more difficult [27].

Besides using the Hamming distance as the fitness function for digital circuits design, there are other possibilities for particular applications. For example, digital image filters can be designed by means of CGP. In this case, the functional specification is not complete, the quality of the candidate circuits is evaluated on a limited training data set [57].

Other applications of CGP include the design and optimization of digital circuits at the transistor level [43], evolutionary design of polymorphic circuits [47] or transistor-level design and optimization of FPGA architectures with respect to production variability [62].

### 2.2.3 Multi-Objective EAs

Unlike the single-objective optimization, which enables to compare any two candidate solutions and decide which one is better, the multi-objective optimization leads to the existence of a set of solutions showing different trade-offs, if the objectives are conflicting.

A multi-objective evolutionary optimization problem can be defined as

$$\begin{array}{lll} \text{minimize/maximize} & f_m(p), & m = 1, 2, ..., M, \\ \text{subject to} & g_j(p) \geq 0, & j = 1, 2, ..., J, \\ & h_k(p) = 0, & k = 1, 2, ..., K, \end{array} \tag{2.9}$$

where $f_m$ are the optimized objectives, $p$ is an individual. The solutions must fulfill the inequity constraints $g_j$ and equity constraints $h_k$ to be acceptable [12].

Many multi-objective evolutionary algorithms have been proposed. Most of them are based on the idea of *Pareto dominance*. The solution $p$ *dominates* the solution $q$ ($p \prec q$) if $p$ is no worse than $q$ in all objectives and $p$ is strictly better than $q$ in at least one objective. The principle can be seen in Figure 2.2, where the Pareto optimal solutions are not dominated by any other solutions and form the so called *Pareto front*.

### Strength Pareto Evolutionary Algorithm 2

Strength Pareto Evolutionary Algorithm 2 (SPEA2), a multi-objective EA introduced by Zitzler et al. [84], maintains two sets of individuals: an archive with non-dominated solutions and a breeding population. In each generation, the fitness of all individuals from both sets is evaluated and the non-dominated solutions are found. The archive is then updated with the non-dominated solutions; a nearest neighbor density estimation algorithm is applied if the archive size is exceeded. The fitness of an individual is computed based on the number of individuals it dominates, the number of individuals that are dominated and the density estimate. The offspring population is created using recombination and mutation of individuals selected using a binary tournament selection [34].

### TSPEA2

TSPEA2 is a branch of SPEA2 introduces by Kaufmann and Platzner [33]. The only difference between SPEA2 and TSPEA2 is that TSPEA2 favours one (main) objective over several others. In the binary tournament, the main objective is first checked and if one of the individuals is better, it is preferred regardless of other objectives. TSPEA2 was motivated by an earlier algorithm MO-Turtle GA introduced by Trefzer et al. [63].

### $\mu$GA and $\mu$GAII

The $\mu$GA [9] and $\mu$GAII [61] algorithms use three populations: an external population for non-dominated individuals of high diversity, a working (breeding) population and an immutable population containing randomized solutions. In each generation, a small set of individuals is selected randomly from the breeding and the immutable population and a standard GA is applied on them. After reaching nominal convergence (the situation when all individuals have similar chromosomes), the best individuals are copied to the breeding and the external population. After several generations, a subset of the breeding population is replaced by non-dominated individuals from the external population [34].

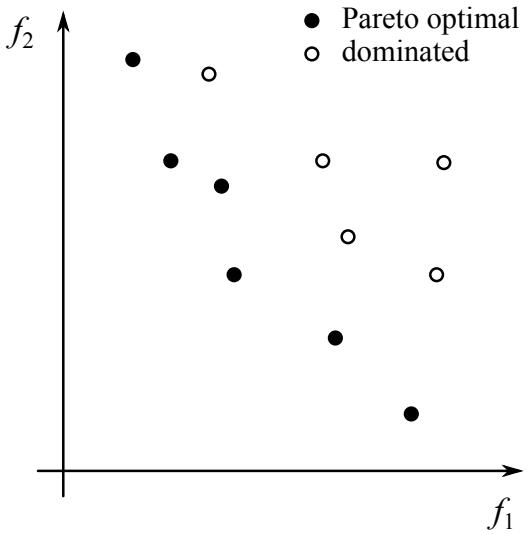



Figure 2.2: Pareto optimal and dominated solutions (when $f_1$ and $f_2$ have to be minimized).

### Non-dominated Sorting Genetic Algorithm II

One of the most popular multi-objective evolutionary algorithms is the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [11]. It is based on sorting individuals from population $P$ according to the dominance relation into multiple fronts. The first front $F_0$ contains all non-dominated 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 (see Algorithm 1) 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 set $S_p$ contains all individuals from the population that are dominated by $p$. The number of individuals that dominate $p$ is denoted by $n_p$. The rank of an individual $p$ (the order of the frontier it belongs to) is denoted by $p_{\text{rank}}$.

fast-non-dominated-sort($P$)
$F_0 = \emptyset$
**foreach** $p \in P$ **do**
  $S_p = \emptyset$
  $n_p = 0$
  **foreach** $q \in P$ **do**
    **if** $p \prec q$ **then**
      $S_p = S_p \cup \{q\}$
    **end**
    **else if** $q \prec p$ **then**
      $n_p = n_p + 1$
    **end**
  **end**
  **if** $n_p = 0$ **then**
    $p_{\text{rank}} = 0$
    $F_0 = F_0 \cup \{p\}$
  **end**
**end**
$i = 0$
**while** $F_i \neq \emptyset$ **do**
  $Q = \emptyset$
  **foreach** $p \in F_i$ **do**
    **foreach** $q \in S_p$ **do**
      $n_q = n_q - 1$
      **if** $n_q = 0$ **then**
        $q_{\text{rank}} = i + 1$
        $Q = Q \cup \{q\}$
      **end**
    **end**
  **end**
  $i = i + 1$
  $F_i = Q$
**end**
$F = (F_0, F_1, \dots)$
**return** $F$

**Algorithm 1:** Non-dominated sort [25].

crowding-distance-assignment($F_i$)
$l = |F_i|$
**foreach** $p \in P$ **do**
  $p_{\text{dist}} = 0$
**end**
**foreach** *objective* $m$ **do**
  $I = \text{sort}(F_i, m)$
  $I[0]_{\text{dist}} = \infty$
  $I[l-1]_{\text{dist}} = \infty$
  **for** $i$ *in* $1$ *to* $l-2$ **do**
    $I[i]_{\text{dist}} = I[i]_{\text{dist}} + \frac{I[i+1]_m - I[i-1]_m}{f_m^{\max} - f_m^{\min}}$
  **end**
**end**

**Algorithm 2:** Crowding distance assignment [25].

constraint-violation-assignment($P$)
**foreach** $p \in P$ **do**
  $p_{\text{constr\_viol}} = 0$
  **foreach** *objective* $m$ **do**
    **if** $p_m < c_m^{\min}$ **then**
      $p_{\text{constr\_viol}} = p_{\text{constr\_viol}} + \frac{c_m^{\min} - p_m}{f_m^{\max}}$
    **end**
    **if** $p_m > c_m^{\max}$ **then**
      $p_{\text{constr\_viol}} = p_{\text{constr\_viol}} + \frac{p_m - c_m^{\max}}{f_m^{\max}}$
    **end**
  **end**
**end**

**Algorithm 3:** Constraint violation assignment [25].

The solutions within the individual fronts are then sorted according to the *crowding distance metric*. This metric helps to preserve the diversity of the population along the fronts [11]. It is computed as the average distance of two solutions on either side along each of the objectives. Solutions on the boundaries are assigned an infinite crowding distance, which ensures that these solutions will always dominate the other solutions (see

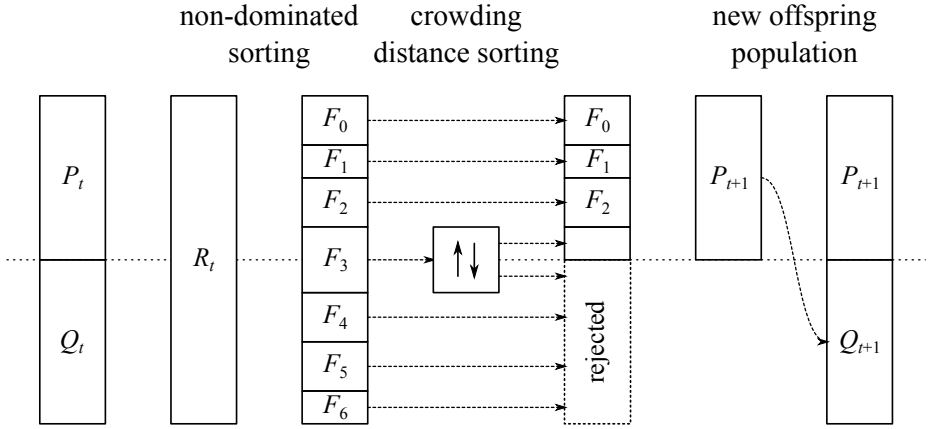



Figure 2.3: NSGA-II algorithm scheme.

Algorithm 2). 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 [25].

Many real world applications require constraining the solutions on particular objectives. NSGA-II offers a simple way to handle the constraints, whilst the low algorithm complexity is preserved. Each solution can be either feasible or infeasible. The infeasible solutions are assigned a constraint violation according to the Algorithm 3. The constraints on the objective $m$ are denoted by $\langle c_m^{\min}, c_m^{\max} \rangle$. When comparing two solutions, a feasible solution is always preferred. If both solutions are infeasible, the solution with smaller constraint violation is better. In the opposite case, if both solutions are feasible, the dominance depends on the rank and the crowding distance metric [25].

The overall algorithm works as follows. In each generation $t$, the parental population $P_t$ and the offspring population $Q_t$ (both of the same size) form an unified population $R_t$. The individuals in $R_t$ are assigned the equivalence rank and the crowding distance. Then, the Pareto fronts are identified and the new parental population $P_{t+1}$ is filled with the individuals from the first fronts as long as $P_{t+1}$ is not overcrowded. The individuals from the last used Pareto front are sorted using the crowding distance and a fraction of them is selected just to fill the population $P_{t+1}$ (see Figure 2.3) [25].

The first attempts to use NSGA-II with CGP used the GA representation of the individuals [34]. Knieper et al. compared the performance of four multi-objective EAs (SPEA2, TSPEA2, NSGA-II and $\mu$GA) with standard GA in the task of combinational 2- and 3-bit adders and multipliers and 6- and 7-parity circuits. Hilder et al. [21] used NSGA-II with CGP to evolve 2- and 3-bit combinational adders and multipliers and a Hex to 7-Segment display driver. Unfortunately, the complexity of the circuits used for the evaluation is not comparable to real world applications in both cases.

Petrlik [48] evolved approximate multiple constant multipliers with respect to multiple objectives by means of NSGA-II and CGP on functional level.

### 2.2.4 Design Acceleration

The evolutionary design is a very computationally demanding approach. In order to reduce the design time, one has to deal with the acceleration of the fitness function or search algorithm modifications. We will briefly survey relevant approaches in the context of CGP.

### Spatially Structured EAs

Spatially structured evolutionary algorithms have been intensively studied in the past and a variety of approaches differing in the used evolutionary algorithm or communication topology has emerged [2, 60]. By introducing multiple populations evolving in parallel, one can increase the population diversity and thus make the EA more explorative leading to a higher probability of finding the global optimum for particular problems.

As the combinational circuit design is a very complex problem, the search space is generally rugged containing lots of local optima and thus the potential of exploiting parallel EA is high. Unfortunately, the absence of a crossover operator in CGP is a very limiting factor since most parallel models take advantage of combining genotypes from different isolated populations. Nevertheless, the model of isolated islands with migration of the best individuals in each population can be applied to CGP [27].

### Coevolutionary Algorithms

Often, the fitness in CGP is calculated over a set of *fitness cases* (e.g. when designing digital image filters) [66]. A fitness case corresponds to a representative situation in which the ability of a program to solve a problem can be evaluated. Each fitness case consists of potential program inputs and target values expected from a perfect solution as a response for these program inputs.

A set of fitness cases can be either a complete specification or just a small sample of the entire domain space. The choice of how many fitness cases (and which ones) to use is often crucial since whether or not an evolved solution will generalize over the entire domain depends on this choice. However, in the case of digital circuit evolution, it is necessary to verify whether a candidate $n$-input circuit generates correct responses for all possible fitness cases (input combinations, i.e. $2^n$ assignments). It was shown that testing just a subset of $2^n$ fitness cases does not lead to correctly working circuits [29].

Hillis [22] introduced an approach that can automatically evolve subsets of fitness cases concurrently with problem solution. He used a two-population coevolutionary algorithm (CoEA) in the task of minimal sorting network design. Subsets of test cases used to evaluate sorting networks evolved simultaneously with the sorting networks. Evolved sorting networks were used to evaluate the test cases subsets. The fitness of each sorting network was measured by its ability to correctly solve fitness cases while the fitness of the fitness cases subsets was better for those that could not be solved well by currently evolved sorting networks. This approach was recently used to evolve digital image filters [28].

Other CoEA approaches and techniques include compositional coevolution [14], indirectly encoded fitness predictors [13] or plastic fitness predictors [81].

Coevolutionary algorithms are traditionally used to evolve interactive behavior which is difficult to evolve with an absolute fitness function. The state of the art of coevolutionary algorithms has recently been summarized in [51].

### Parallelization

When designing combinational circuits, the CGP implementation usually must process all the $2^{n_\mathrm{i}}$ test vectors on the whole phenotype for the entire population of individuals and compare all the $n_\mathrm{o}$ outputs to the desired ones. In order to take advantage of modern superscalar out-of-order processors, the parallelism at various levels has to be employed and special attention to memory access policy has to be paid.

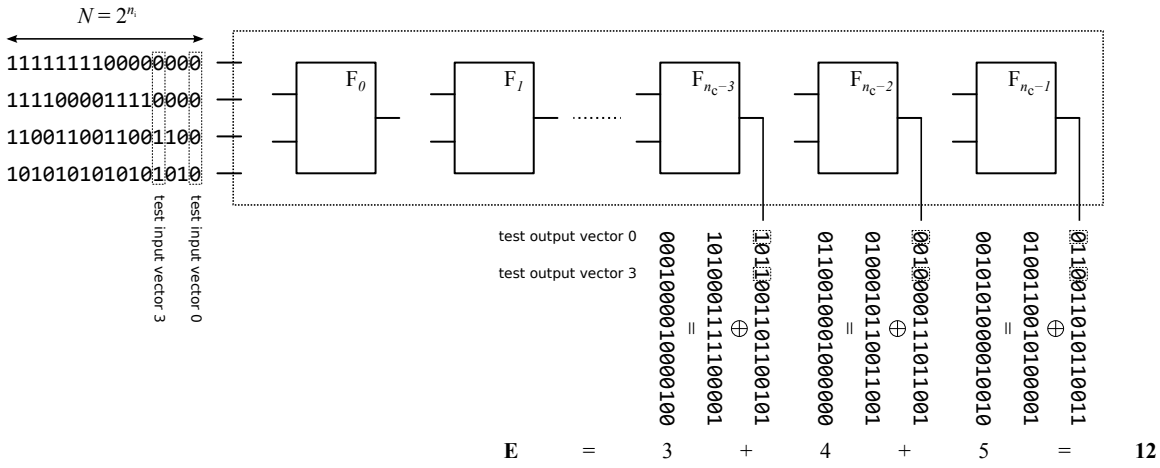



Figure 2.4: Parallel evaluation of a CGP individual – multiple test vectors can be evaluated in parallel using bitwise or vector operations. The hamming distance can be efficiently calculated by XORing the output value with the desired one and counting the number of ones.

- *Vectorization*: The most fundamental optimization we can apply is the bit-level parallelism. Instead of separate test vector processing, up to 64 test vectors can be processed in parallel on 64-bit processors thanks to bitwise operations (see Figure 2.4). Furthermore, by introducing the data-level parallelism using SIMD instructions, 128, 256 or even 512 test vectors can fit into the SSE, AVX or AVX-512 registers respectively.

  Significant speed-up can be achieved by introducing the so called native implementation [73, 27]. Instead of traversing the chromosome and computing the node outputs directly, the chromosome is compiled at first. The compiled program is then executed on each test vector for each individual in the population.

- *Thread Parallelism*: The most straightforward way of dividing the computations into multiple threads is to assign each thread a subset of the population and compute the fitness values in parallel. However, CGP uses a very small population, often much smaller than the number of physical cores present in today's processors. Nevertheless, one can parallelize the fitness function in a different way, such as assigning each thread a portion of the test vectors [27].

- *General Purpose GPUs*: Recent advances in scientific computing have made it possible to use general purpose GPUs (GPGPUs) for parallel EAs. GPGPUs are low-cost, massively parallel, many-core processors. Although the parallelism of EAs is well suited for the single-program multiple-data based GPGPUs, there are many issues to be resolved such as the thread divergence caused by the randomness of EAs. The state of the art of EAs on GPGPUs has been recently summarized in [64].

- *Coprocessors*: Coprocessors have been mostly used to accelerate specific tasks, e.g. audio or video encoding/decoding, cryptography etc. Recently, Intel introduced a general purpose Many Integrated Core Architecture (Intel MIC). Intel Xeon Phi coprocessor is an example of this approach, it has been designed for applications that can exploit vector instructions and are scalable enough to efficiently run in a huge number

of threads [30]. Unlike GPGPUs, the user can exploit standard programming model and thus reuse a lot of code optimized for CPUs. However, to reach the maximum performance, one has to seriously deal with manual code optimizations [24].

- *Computer Clusters*: Spatially structured EAs are inherently suitable for running on computer clusters. The distributive nature of spatially structured EAs in combination with other complementary parallelization techniques enables to fully utilize multiple computing nodes. Communication is usually not a bottleneck, since the populations are evolving on individual nodes independently and exchange data occasionally [26, 27].

#### Hardware Accelerators

Reconfigurable hardware (i.e. FPGAs or hybrid platforms, such as Xilinx Zynq) offers a great possibility for accelerating computationally intensive applications. Recently, CGP has been accelerated by means of so called Virtual Reconfigurable Circuit (VRC) [55] or Dynamic Partial Reconfiguration (DPR) [53]. Multiple VRCs have been used to even increase the performance [28, 70].

#### Formal Methods

Computing the fitness function for complex digital circuits (i.e. circuits with more than 20 inputs) is not efficient. In the case of evolutionary optimization, the exact fitness value is often not needed, because the evolution starts with a fully working circuit and every destructive mutation is unwanted. Therefore, checking the output equivalence of the original and the candidate circuit is sufficient to perform in this case.

Recently, the fitness calculation has been sped up by introducing formal methods, e.g. based on the Boolean Satisfiability (SAT) problem [69] or the Binary Decision Diagrams (BDD) [71]. Although the fitness function is mostly based on Hamming distance [71], the latest published results suggest that formal methods can be used to calculate even more complex error metrics (e.g. the worst case error) [23].

- *SAT Solvers*: The problem of output equivalence can be easily transformed to the Boolean satisfiability problem, which can be then solved by means of standard tools (SAT solvers) [69].

- *Binary Decision Diagrams*: A BDD is a directed acyclic graph with one root and two terminal nodes that are referred to as '0' and '1'. The other (non-terminal) nodes are associated with a primary input variable and have exactly two outgoing edges corresponding to assigning the variable *true* or *false* truth value. Every path in a BDD is unique; if we find a path from the root node to the terminal node '1', then we have found a value assignment to the variables for which the function is evaluated to 1. A CGP individual can be represented by a BDD. When properly used, various error metrics can be computed [23].

# Chapter 3

# Research Summary

This chapter summarizes the research presented in the thesis. After a brief overview of the research process, the motivation and abstracts for each included paper are presented. Finally, a complete list of publications, research projects, grants and awards are listed.

## 3.1 Overview

The research presented in this thesis extends the previous research in several ways. The scalability of the evolutionary design method is improved by introducing a highly optimized parallel implementation of CGP. To address all demands placed by the hardware community, the method is extended to be multi-objective and the estimation accuracy of various circuit parameters is substantially improved. The thesis primarily deals with approximate circuits design, the performance of the method is demonstrated on several real-world problems.

The research started with a detailed analysis of the state of the art methods. It was shown in Chapter 2 that the evolutionary design methods suffer from low scalability and thus, a highly optimized CGP implementation was proposed and various acceleration techniques were analyzed in Paper I. The scalability of the implementation was then evaluated on several problems – design of combinational adders and multipliers (Paper I) and bent Boolean functions (Paper II). The design of the bent Boolean functions was a very complex problem with a high potential of parallelization and thus the Xeon Phi Coprocessor was utilized to further accelerate the design process (Paper III).

The work was then directed to the multi-objective design approach. The first version of multi-objective CGP was published in Paper IV. In Paper V, the method was improved by replacing the randomly generated initial population by a set of conventional circuits. The accuracy of the estimation of circuit parameters was enhanced by more accurate modeling of a real technology process library. The results were compared to a state of the art method and published as the EvoApprox8b library in Paper VII.

In Paper VI, the method was used to generate approximate circuits to be used in a TMR schema. Experimental results demonstrated that the evolutionary approach produced better solutions than the probabilistic approach developed by our colleagues from University Carlos III de Madrid.

## 3.2 Papers included in this thesis

### 3.2.1 Paper I

Radek Hrbáček and Lukáš Sekanina. **Towards Highly Optimized Cartesian Genetic Programming: From Sequential via SIMD and Thread to Massive Parallel Implementation.** In *GECCO '14 Proceedings of the 2014 conference on Genetic and evolutionary computation.* New York: Association for Computing Machinery, 2014, pp. 1015-1022. ISBN 978-1-4503-2662-9.

Author participation: 80 %.
Conference Rank: A1 (Qualis).

**Abstract**

Most implementations of Cartesian genetic programming (CGP) which can be found in the literature are sequential. However, solving complex design problems by means of genetic programming requires parallel implementations of search methods and fitness functions. This paper deals with the design of highly optimized implementations of CGP and their detailed evaluation in the task of evolutionary circuit design. Several sequential implementations of CGP have been analyzed and the effect of various additional optimizations has been investigated. Furthermore, the parallelism at the instruction, data, thread and process level has been applied in order to take advantage of modern processor architectures and computer clusters. Combinational adders and multipliers have been chosen to give a performance comparison with state of the art methods.

**Contribution**

As a highly optimized implementation of the evolutionary design method based on CGP was one of the first goals of the research, a deep analysis of possible optimization techniques was desirable. Such an analysis is covered within this paper. Although the results presented in this paper suggest that the most efficient approach, at least for complex circuits, is the native implementation (see Chapter 2), subsequent research revealed a weakness of this method – low flexibility in terms of function set modification. Therefore, the standard interpreted approach was preferred in further research.

*This work resulted in a very efficient CGP implementation capable of running on a wide range of computers – from single-core processors to supercomputers.*

### 3.2.2 Paper II

Radek Hrbáček and Václav Dvořák. **Bent Function Synthesis by Means of Cartesian Genetic Programming.** In *Parallel Problem Solving from Nature - PPSN XIII.* Heidelberg: Springer Verlag, 2014, LNCS vol. 8672, pp. 414-423. ISBN 978-3-319-10761-5.

Author participation: 80 %.
Conference Rank: A2 (Qualis).

**Abstract**

In this paper, a new approach to synthesize bent Boolean functions by means of Cartesian Genetic Programming (CGP) is proposed. Bent functions have important applications in

cryptography due to their high nonlinearity. However, they are very rare and their discovery using conventional brute force methods is not efficient enough. We show that by using CGP we can routinely design bent functions of up to 16 variables. The evolutionary approach exploits parallelism in both the fitness calculation and the search algorithm.

**Contribution**

The proposed efficient CGP implementation presented in the previous papers can be applied to a wide range of applications. In this paper, the method is used to find Boolean functions with the highest nonlinearity, which are very rare, but very important for networking and cryptography.

*This work resulted in a new efficient approach for finding Boolean functions with high nonlinearity. It was the first time CGP was successfully used for this purpose and the paper gave an impulse for other researchers to further develop this approach [50, 49]. The results were awarded by the bronze medal at the Humies competition (2014).*

### 3.2.3 Paper III

Radek Hrbáček. **Bent Functions Synthesis on Xeon Phi Coprocessor.** In *Mathematical and Engineering Methods in Computer Science*. Heidelberg: Springer Verlag, 2014, LNCS vol. 8934, pp. 88-99. ISBN 978-3-319-14895-3.

Author participation: 100 %.

**Abstract**

A new approach to synthesize bent Boolean functions by means of Cartesian Genetic Programming (CGP) has been proposed recently. Bent functions have important applications in cryptography due to their high nonlinearity. However, they are very rare and their discovery using conventional brute force methods is not efficient enough. In this paper, a new parallel implementation is proposed and the performance is evaluated on the Intel Xeon Phi Coprocessor.

**Contribution**

The computational demands of the method proposed in Paper II are very high. The fitness (nonlinearity) evaluation time grows exponentially with the number of variables. However, there is a great potential of parallelization, even higher than in the case of the fitness function based on Hamming distance. This paper deals with the implementation and optimization of the method for running on the Intel Xeon Phi Coprocessor. The implementation is highly parallel and allows to utilize all 60 cores of the coprocessor by running 240 threads.

*This work resulted in a significant speedup and an increase in complexity of the bent functions designed using the proposed evolutionary design method – up to 18 variable bent functions were found.*

### 3.2.4 Paper IV

Radek Hrbáček. **Parallel Multi-Objective Evolutionary Design of Approximate Circuits.** In *GECCO '15 Proceedings of the 2015 conference on Genetic and evolutionary*

*computation*. New York: Association for Computing Machinery, 2015, pp. 687-694. ISBN 978-1-4503-3472-3.

Author participation: 100 %.
Conference Rank: A1 (Qualis).

**Abstract**

Evolutionary design of digital circuits has been well established in recent years. Besides correct functionality, the demands placed on current circuits include the area of the circuit and its power consumption. By relaxing the functionality requirement, one can obtain more efficient circuits in terms of the area or power consumption at the cost of an error introduced to the output of the circuit. As a result, a variety of trade-offs between error and efficiency can be found. In this paper, a multi-objective evolutionary algorithm for the design of approximate digital circuits is proposed. The scalability of the evolutionary design has been recently improved using parallel implementation of the fitness function and by employing spatially structured evolutionary algorithms. The proposed multi-objective approach uses Cartesian Genetic Programming for the circuit representation and a modified NSGA-II algorithm. Multiple isolated islands are evolving in parallel and the populations are periodically merged and new populations are distributed across the islands. The method is evaluated in the task of approximate arithmetical circuits design.

**Contribution**

Since the most important goal of the thesis was to develop an automated design method capable of multi-objective evolutionary design, an extension to the standard CGP was needed. This paper introduces such an extension. The approach is based on the NSGA-II algorithm, but several modifications were required to adapt the algorithm for CGP. The implementation preserves all benefits of the single-objective parallel CGP implementation – even a new multi-objective island model was introduced to utilize computer clusters.

*As a result of this work, the CGP implementation was extended with the multi-objective approach. The method was used to design approximate arithmetical circuits from scratch.*

### 3.2.5 Paper V

Radek Hrbáček, Vojtěch Mrázek and Zdeněk Vašíček. **Automatic Design of Approximate Circuits by Means of Multi-Objective Evolutionary Algorithms.** In *Proceedings of the 11th International Conference on Design & Technology of Integrated Systems in Nanoscale Era*. Istanbul: Istanbul Sehir University, 2016, pp. 239-244. ISBN 978-1-5090-0335-8.

Author participation: 50 %.

**Abstract**

Recently, power efficiency has become the most important parameter of many real circuits. At the same time, a wide range of applications capable of tolerating imperfections has spread out especially in multimedia. Approximate computing, an emerging paradigm, takes advantage of relaxed functional requirements to make computer systems more efficient in terms of energy consumption, speed or complexity. As a result, a variety of trade-offs

between error and efficiency can be found. In this paper, a design method based on a multi-objective evolutionary algorithm is proposed. For a given circuit, the method is able to produce a set of Pareto optimal solutions in terms of the error, power consumption and delay. The proposed design method uses Cartesian Genetic Programming for the circuit representation and a modified NSGA-II algorithm for design space exploration. The method is used to design Pareto optimal approximate versions of arithmetic circuits such as multipliers and adders.

**Contribution**

In Paper IV, a new multi-objective CGP implementation was proposed. However, the estimation of circuit parameters was simplified and the complexity of evolved circuits was relatively low (4-bit combinational adders and multipliers). In this paper, we proposed to use a set of conventional arithmetic circuits of various architectures as the initial population instead of randomly generated initial population. Moreover, we implemented more accurate estimation of the power consumption, propagation delay and area of the circuits based on an existing technology process library.

*As a result of this work, hundreds of approximate 8-bit adders and multipliers were designed with respect to three objectives – mean relative error, power consumption and delay.*

### 3.2.6 Paper VI

Antonio José Sánchez-Clemente, Luis Entrena, Radek Hrbáček and Lukáš Sekanina. **Error Mitigation using Approximate Logic Circuits: A Comparison of Probabilistic and Evolutionary Approaches.** *IEEE Transactions on Reliability.* 2016, vol. 65, no. 4, pp. 1871-1883. ISSN 0018-9529.

Author participation: 25 %.
Impact factor: 2.287.

**Abstract**

Technology scaling poses an increasing challenge to the reliability of digital circuits. Hardware redundancy solutions, such as Triple Modular Redundancy, produce very high area overhead, so partial redundancy is often used to reduce the overheads. Approximate logic circuits provide a general framework for optimized mitigation of errors arising from a broad class of failure mechanisms, including transient, intermittent and permanent failures. However, generating an optimal redundant logic circuit that is able to mask the faults with the highest probability while minimizing the area overheads is a challenging problem. In this work we propose and compare two new approaches to generate approximate logic circuits to be used in a TMR schema. The probabilistic approach approximates a circuit in a greedy manner based on a probabilistic estimation of the error. The evolutionary approach can provide radically different solutions that are hard to reach by other methods. By combining these two approaches, the solution space can be explored in depth. Experimental results demonstrate that the evolutionary approach can produce better solutions, but the probabilistic approach is close. On the other hand, these approaches provide much better scalability than other existing partial redundancy techniques.

**Contribution**

In this work, we provided the CGP based method for the design of approximate logic circuits to be used in partially protected circuits in a TMR schema. The advantage of the proposed technique was the ability to generate good trade-off solutions between the reliability and the area overhead of the circuit.

*This work resulted in a new method of designing approximate logic circuits to be used in a TMR schema. In comparison with other existing partial redundancy techniques, the proposed method provided much better scalability.*

### 3.2.7 Paper VII

Vojtěch Mrázek, Radek Hrbáček, Zdeněk Vašíček and Lukáš Sekanina. **EvoApprox8b: Library of Approximate Adders and Multipliers for Circuit Design and Benchmarking of Approximation Methods.** In *Proc. of the 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Lausanne: European Design and Automation Association, 2017, pp. 258-261. ISBN 978-3-9815370-9-3.

Author participation: 25 %.
Conference Rank: A1 (Qualis).

**Abstract**

Approximate circuits and approximate circuit design methodologies attracted a significant attention of researchers as well as industry in recent years. In order to accelerate the approximate circuit and system design process and to support a fair benchmarking of circuit approximation methods, we propose a library of approximate adders and multipliers called EvoApprox8b. This library contains 430 nondominated 8-bit approximate adders created from 13 conventional adders and 471 non-dominated 8-bit approximate multipliers created from 6 conventional multipliers. These implementations were evolved by a multi-objective Cartesian genetic programming. The EvoApprox8b library provides Verilog, Matlab and C models of all approximate circuits. In addition to standard circuit parameters, the error is given for seven different error metrics. The EvoApprox8b library is available at: www.fit.vutbr.cz/research/groups/ehw/approxlib.

**Contribution**

This paper compares the results of Paper V to a conventional design method and presents the evolved circuits as a library of circuits (synthesized using Synopsys Design Compiler for 180nm and 45nm technologies).

*As a result of this paper, a free library of approximate arithmetic circuits was proposed. The library provides several representations of the circuits along with their properties. One can filter the circuits by the properties. The work was awarded by the Best IP Award at the Design, Automation and Test in Europe (DATE) conference 2017.*

## 3.3 List of Other Publications

- Radek Hrbáček. Simulation Based Neural Motion Planner Learning. In *Proceedings of the 17th Conference STUDENT EEICT 2011 Volume 1.* Brno: NOVPRESS s.r.o.,

2011. pp. 189-191. ISBN: 978-80-214-4271-9.
Author participation: 100 %.

- Radek Hrbáček. Introduction to Compressive Sampling. In *Proceedings of the 17th Conference STUDENT EEICT 2011 Volume 1.* Brno: NOVPRESS s.r.o., 2011. pp. 45-47. ISBN: 978-80-214-4271-9.
  Author participation: 100 %.

- Jan Hrbáček, Radek Hrbáček and Stanislav Věchet. Modular Control System Architecture for a Mobile Robot. In *Proceedings of the 17th international conference Engineering Mechanics 2011.* 1. Prague: Institute of Thermomechanics, Academy of Sciences of the Czech Republic, 2011. pp. 211-214. ISBN: 978-80-87012-33-8.
  Author participation: 33 %.

- Radek Hrbáček, Pavel Rajmic, Vítězslav Veselý and Jan Špiřík. Introduction to sparse signal representations. *Elektrorevue – internet journal (http://www.elektrorevue.cz)*, 2011, vol. 2011, no. 50, pp. 1-10. ISSN: 1213-1539.
  Author participation: 40 %.

- Radek Hrbáček, Pavel Rajmic, Vítězslav Veselý and Jan Špiřík. Sparse signal representations: compressed sensing. *Elektrorevue – internet journal (http://www.elektrorevue.cz)*, 2011, vol. 2011, no. 67, pp. 1-6. ISSN: 1213-1539.
  Author participation: 44 %.

- Jiri Krejsa, Stanislav Věchet, Jan Hrbáček, Tomáš Ripel, Vítězslav Ondroušek, Radek Hrbáček and Petr Schreiber. Presentation robot Advee. *Engineering Mechanics*, 2012, vol. 18, no. 5/6, pp. 307-322. ISSN: 1802-1484.
  Author participation: 5 %.

- Radek Hrbáček. Hardware Platform for Coevolutionary Design. In *Proceedings of the 19th Conference STUDENT EEICT 2013 Volume 2.* Brno: LITERA, 2013. pp. 279-281. ISBN: 978-80-214-4694-6.
  Author participation: 100 %.

- Radek Hrbáček and Michaela Šikulová. Coevolutionary Cartesian Genetic Programming in FPGA. In *Advances in Artificial Life, ECAL 2013, Proceedings of the Twelfth European Conference on the Synthesis and Simulation of Living Systems.* Cambridge, US: MIT, 2013. pp. 431-438. ISBN 978-0-262-31709-2.
  Author participation: 60 %
  Conference Rank: B1 (Qualis).

- Jiří Toman and Radek Hrbáček. Redundant Control Algorithm for a Brushless DC Motor. In *Electrical Drives and Power Eletronics.* Košice: Technical University of Košice, 2015, pp. 117-124. ISBN: 978-80-553-2208-7.
  Author participation: 50 %.

- Filip Vaverka, Radek Hrbáček and Lukáš Sekanina. Evolving Component Library for Approximate High Level Synthesis. In *2016 IEEE Symposium Series on Computational Intelligence.* Athens: IEEE Computational Intelligence Society, 2016, pp. 1-8. ISBN 978-1-5090-4240-1.
  Author participation: 25 %.
  Conference Rank: B5 (Qualis).

## 3.4 Research Projects and Grants

### 3.4.1 Czech Science Foundation

- GA16-17538S *Relaxed equivalence checking for approximate computing.*
  Co-investigator.
- GA14-04197S *Advanced Methods for Evolutionary Design of Complex Digital Circuits.*
  Co-investigator.

### 3.4.2 Anselm & Salomon Supercomputer Allocations

- OPEN-8-4 *Multi-objective Approximate Circuit Design on Computer Cluster.*
  500 000 core hours.
  Primary investigator.
- IT4T-10-4 *Evolutionary Design of Cryptographic Boolean Functions.*
  300 000 core hours.
  Investigator.
- IT4T-9-2 *Approximate circuit design on computer cluster.*
  300 000 core hours.
  Investigator.
- IT4I-7-6 *Evolutionary design on computer cluster.*
  80 000 core hours.
  Investigator.
- IT4I-5-9 *Evolvable hardware on computer cluster.*
  75 000 core hours.
  Investigator.

## 3.5 Awards

- Special Prize IT4Innovations (Joseph Fourier Prize 2017).
- Best Interactive Presentation (DATE 2017).
- Bronze medal in Humies competition (GECCO 2014).
- BUT rector's award for excellent master study and scientific research results.
- 1. prize in the competition *ICT Master thesis of the year 2013* awarded for the master thesis *Coevolutionary Algorithm in FPGA*.
- FIT BUT dean's award for the master thesis *Coevolutionary Algorithm in FPGA*.
- 1. prize in the *Student EEICT 2013* competition awarded for the paper *Hardware Platform for Coevolutionary Design*.

# Chapter 4

# Discussion and Conclusions

This chapter discusses and summarizes the results presented in the thesis and gives conclusions and possible directions for future work.

## 4.1 The Approach

An analysis of the state of the art evolutionary design methods and a study of possible acceleration techniques have been performed within the research presented in this thesis. In order to accomplish the given research objectives, a new highly optimized implementation of Cartesian Genetic Programming was proposed. The next step was to extend the CGP to support the design with respect to multiple objectives – a modified NSGA-II algorithm was used for this purpose. The performance of the implementation was evaluated in multiple different applications, in particular (approximate) combinational arithmetic circuits design, bent Boolean functions discovery, approximate logic circuits for TMR schema and others. The experiments were conducted on computers operated by two organizations – MetaCentrum VO and IT4Innovations (supercomputers Anselm, Salomon). All particular research steps are presented in the description of relevant scientific papers mentioned in Chapter 3.

## 4.2 Software Outcomes

During the research, a new CGP implementation was developed and actively used. This implementation was improved and extended with new features step by step to maintain the universality of the tool as great as possible, except for a few particular cases (e.g. Intel Xeon Phi coprocessor implementation).

The CGP tool is implemented using C/C++. Thread level parallelism is based on the OpenMP library and the island model utilizes MPI message passing communication approach. The tool is a command line utility, where all parameters are passed to the tool using command line arguments. The desired circuit functionality can be specified either as a truth table or by importing a PLA (Programmable Logic Array) file. The evolutionary design can start either from scratch (randomly generated initial population) or a set of CGP chromosomes can be imported. The results of the evolutionary design can be exported to various output formats:

- chromosome file – CGP chromosome representation,
- VHDL, Verilog and BENCH – hardware description languages,

- C/C++ – logic and arithmetic representation.

The CGP implementation is able to model various technology processes as listed in Table 4.1. The simplest one implements general logic *gates* (BUF, NOT, AND, OR, XOR, NAND, NOR, XNOR, 1, 0) and the estimation of the area, power consumption and delay is very rough.

A much more realistic technology library *osu180* is based on a real technology process library. It is available in four variants; from a set of up to 4-input gates to a subset containing just 2-input gates.

The last technology supported by the tool is the *look-up table* (LUT) technology with 3-6 node inputs. The LUT-level implementation was optimized by the tool itself (on gate-level) by generating highly optimized implementations of all 3-input and 4-input Boolean functions. These optimized functions were exported to C/C++ and compiled together into a new LUT-level implementation. This implementation is intended for the design and approximation of digital circuits running on FPGAs. The LUT-based approach is currently prepared for publication.

| **Technology** | **Node inputs** $n_{\mathrm{ni}}$ | **Node outputs** $n_{\mathrm{no}}$ | **Node functions** $n_{\mathrm{f}}$ |
|---|---|---|---|
| *gate* | 2 | 1 | 10 |
| *osu180* | 4 | 2 | 17 |
| | 3 | 2 | 15 |
| | 2 | 2 | 9 |
| | 2 | 1 | 8 |
| *look-up table* | 3 | 1 | 256 |
| | 4 | 1 | 65536 |
| | 5 | 1 | 4294967296 |
| | 6 | 1 | $2^{64}$ |

Table 4.1: List of supported CGP circuit primitives.

All the technologies can be used with the same algorithms. There are basically two different approaches supported by the tool. The *single-objective* algorithm is based on standard CGP. The fitness function can be composed of multiple objectives; when reaching the maximum of the first objective, the next one is included into the fitness value calculation and so on. The *multi-objective* algorithm is an extension of CGP with a modified NSGA-II algorithm. All objectives are optimized simultaneously. Both algorithms support constraining the individual objectives by entering a minimum or/and a maximum value. The supported fitness functions are as follows:

- Hamming distance, on/off-set Hamming distance
- mean absolute error (MAE), mean squared error (MSE), mean relative error (MRE)
- worst case error (WCE), worst case relative error (WCRE)
- variance of absolute error (VAE)
- error probability (EP)
- area, delay, power, power-delay product (PDP)

Both, single-objective and multi-objective algorithms can be executed on multiple isolated islands. On each island, a separate population of individuals is evolving independently of the others until a predefined number of generations or a time interval is reached. In the single-objective case, the best individual of all islands is determined and the evolutionary process is restarted on all islands with the best individual as a seed. In the multi-objective case, the populations of all islands are merged, a global Pareto front is formed and the individual populations are seeded with individuals from the front.

## 4.3 Contributions

The main contributions of this thesis can be summarized as follows:

- Development of a new highly optimized parallel implementation of CGP that is much more scalable than the state of the art implementations and thus applicable for complex problems.
- Extension of CGP with new function sets based on real technology processes (180nm CMOS library, LUTs).
- Extension of CGP with multi-objective design capability.
- Evolved comprehensive library of 8-bit approximate combinational adders and multipliers that can be used for benchmarking of approximate design methods or as a component library for high-level approximate circuit design.
- Evolved bent Boolean functions of up to 18 variables that are more difficult instances than previous solutions.
- Evolved approximate combinational circuits for TMR schema that show better properties than the circuits obtained by a conventional probabilistic method.

The aforementioned contributions are important for approximate computing as well as for other research areas dealing with digital circuits or related technologies. The work has been awarded by both EAs (Bronze medal in Humies competition) and hardware (DATE Best IP Award) community.

## 4.4 Future Work

The scalability of the CGP method was significantly improved, as presented in this thesis. However, a great potential in further increasing the complexity of problems solved by CGP resides in using formal verification methods in the fitness function. Until recently, these methods could only be used to check if two circuits are equivalent in terms of their output responses. The latest results suggest that some formal methods (e.g. binary decision diagrams) could be used to directly determine an arithmetic distance or to check if a circuit satisfies an arithmetic constraint [23]. Integrating these methods to the proposed CGP implementation is straightforward – only the fitness function has to be re-designed.

The EvoApprox8b library presented in this thesis can be directly used as a component library for a high level approximate circuit synthesis algorithm. For example, a multi-objective evolutionary algorithm has been developed for this purpose [74]. This way, the scalability of the method can be further improved.

The LUT-level CGP implementation mentioned in this chapter promises to further extend the application area of the proposed method to FPGA devices. This will need further experiments and it will be definitely an interesting and challenging continuation of the research started in this thesis.

Another issue that should also be addressed is high computation time of the method in comparison with conventional synthesis methods. Nevertheless, the evolutionary design method is still valuable considering the fact that better results can be obtained in comparison with the conventional methods. Besides, the performance and parallelism of computers is growing and the EA based methods can take advantage of that, as shown in this thesis.

# Bibliography


[1] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford, UK: Oxford University Press, 1996. ISBN: 0-19-509971-0.

[2] E. Cantu-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Norwell, MA, USA: Kluwer Academic Publishers, 2000. ISBN: 0792372212.

[3] M. Carbin, S. Misailovic, and M. C. Rinard. Verifying Quantitative Reliability for Programs That Execute on Unreliable Hardware. In: *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*. OOPSLA '13. Indianapolis, Indiana, USA: ACM, 2013, pp. 33–52. ISBN: 978-1-4503-2374-1. DOI: 10.1145/2509136.2509546. URL: http://doi.acm.org/10.1145/2509136.2509546.

[4] P. Cattani and C. Johnson. ME-CGP: Multi Expression Cartesian Genetic Programming. In: *Evolutionary Computation (CEC), 2010 IEEE Congress on*. 2010, pp. 1–6. DOI: 10.1109/CEC.2010.5586478.

[5] A. Chandrakasan and R. Brodersen. Minimizing power consumption in digital CMOS circuits. In: *Proceedings of the IEEE* 83.4 (1995), pp. 498–523. ISSN: 0018-9219. DOI: 10.1109/5.371964.

[6] V. Chippa et al. Analysis and characterization of inherent application resilience for approximate computing. In: *Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE*. 2013, pp. 1–9.

[7] M. D. Ciletti. *Advanced Digital Design with the Verilog HDL*. 2nd. Upper Saddle River, NJ, USA: Prentice Hall Press, 2010. ISBN: 0136019285, 9780136019282.

[8] J. Clegg, J. A. Walker, and J. F. Miller. A new crossover technique for Cartesian genetic programming. In: *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*. Vol. 2. London: ACM Press, 2007, pp. 1580–1587. URL: http://doi.acm.org/10.1145/1276958.1277276.

[9] C. A. C. Coello and G. T. Pulido. A Micro-Genetic Algorithm for Multiobjective Optimization. In: *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*. EMO '01. London, UK, UK: Springer-Verlag, 2001, pp. 126–140. ISBN: 3-540-41745-1. URL: http://dl.acm.org/citation.cfm?id=647889.739252.

[10] J. Cong and K. Minkovich. Optimality Study of Logic Synthesis for LUT-based FPGAs. In: *Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays*. FPGA '06. Monterey, California, USA: ACM, 2006, pp. 33–40. ISBN: 1-59593-292-5. DOI: 10.1145/1117201.1117207. URL: http://doi.acm.org/10.1145/1117201.1117207.

[11] K. Deb et al. *A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II*. Piscataway, NJ, USA, Apr. 2002. DOI: 10.1109/4235.996017. URL: http://dx.doi.org/10.1109/4235.996017.

[12] K. Deb and D. Kalyanmoy. *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, NY, USA: John Wiley & Sons, Inc., 2001. ISBN: 047187339X.

[13] M. Drahošová, J. Hulva, and L. Sekanina. Indirectly Encoded Fitness Predictors Coevolved with Cartesian Programs. In: *Genetic Programming*. LNCS 9025. Berlin, DE: Springer International Publishing, 2015, pp. 113–125. ISBN: 978-3-319-16500-4. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=10775.

[14] M. Drahošová, G. Komjáthy, and L. Sekanina. Towards Compositional Coevolution in Evolutionary Circuit Design. In: *2014 IEEE International Conference on Evolvable Systems Proceedings*. Piscataway, US: Institute of Electrical and Electronics Engineers, 2014, pp. 157–164. ISBN: 978-1-4799-4479-8. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=10655.

[15] V. Gupta et al. Low-Power Digital Signal Processing Using Approximate Adders. In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 32.1 (2013), pp. 124–137. ISSN: 0278-0070. DOI: 10.1109/TCAD.2012.2217962.

[16] J. Han and M. Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In: *Test Symposium (ETS), 2013 18th IEEE European*. 2013, pp. 1–6. DOI: 10.1109/ETS.2013.6569370.

[17] S. Harding, J. Miller, and W. Banzhaf. Self modifying Cartesian Genetic Programming: Parity. In: *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*. 2009, pp. 285–292. DOI: 10.1109/CEC.2009.4982960.

[18] S. Harding, J. F. Miller, and W. Banzhaf. Self Modifying Cartesian Genetic Programming: Fibonacci, Squares, Regression and Summing. In: *Proceedings of the 12th European Conference on Genetic Programming*. EuroGP '09. Tübingen, Germany: Springer-Verlag, 2009, pp. 133–144. ISBN: 978-3-642-01180-1. DOI: 10.1007/978-3-642-01181-8_12. URL: http://dx.doi.org/10.1007/978-3-642-01181-8_12.

[19] S. Harding, J. Miller, and W. Banzhaf. Developments in Cartesian Genetic Programming: self-modifying CGP. English. In: *Genetic Programming and Evolvable Machines* 11.3-4 (2010), pp. 397–439. ISSN: 1389-2576. DOI: 10.1007/s10710-010-9114-1. URL: http://dx.doi.org/10.1007/s10710-010-9114-1.

[20] S. L. Harding, J. F. Miller, and W. Banzhaf. Self-modifying cartesian genetic programming. In: *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*. Vol. 1. London: ACM Press, 2007, pp. 1021–1028. DOI: doi:10.1145/1276958.1277161. URL: http://www.cs.bham.ac.uk/~wbl/biblio/gecco2007/docs/p1021.pdf.

[21] J. Hilder, J. Walker, and A. Tyrrell. Use of a multi-objective fitness function to improve cartesian genetic programming circuits. In: *Adaptive Hardware and Systems (AHS), 2010 NASA/ESA Conference on*. 2010, pp. 179–185. DOI: 10.1109/AHS.2010.5546262.

[22] W. D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In: *Physica D* 42.1 (1990), pp. 228–234.

[23] L. Holík et al. Towards Formal Relaxed Equivalence Checking in Approximate Computing Methodology. In: *2nd Workshop on Approximate Computing (WAPCO 2016)*. Prague, 2016, pp. 1–6. URL: http://www.fit.vutbr.cz/research/view_pub.php.en?id=11008.

[24] R. Hrbacek. Bent Functions Synthesis on Xeon Phi Coprocessor. In: *Mathematical and Engineering Methods in Computer Science*. LNCS 8934. Heidelberg, DE: Springer Verlag, 2014, pp. 88–99. ISBN: 978-3-319-14895-3. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=10720.

[25] R. Hrbacek. Parallel Multi-Objective Evolutionary Design of Approximate Circuits. In: *GECCO '15 Proceedings of the 2015 conference on Genetic and evolutionary computation*. Under review. New York, US: Association for Computing Machinery, 2015, p. 8. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=10815.

[26] R. Hrbacek and V. Dvorak. Bent Function Synthesis by Means of Cartesian Genetic Programming. In: *Parallel Problem Solving from Nature – PPSN XIII*. Ed. by T. Bartz-Beielstein et al. Vol. 8672. Lecture Notes in Computer Science. Heidelberg: Springer, 2014, pp. 414–423. ISBN: 978-3-319-10761-5. DOI: 10.1007/978-3-319-10762-2_41.

[27] R. Hrbacek and L. Sekanina. Towards Highly Optimized Cartesian Genetic Programming: From Sequential via SIMD and Thread to Massive Parallel Implementation. In: *GECCO '14 Proceedings of the 2014 conference on Genetic and evolutionary computation*. New York, US: Association for Computing Machinery, 2014, pp. 1015–1022. ISBN: 978-1-4503-2662-9.

[28] R. Hrbacek and M. Sikulova. Coevolutionary Cartesian Genetic Programming in FPGA. In: *Advances in Artificial Life, ECAL 2013, Proceedings of the Twelfth European Conference on the Synthesis and Simulation of Living Systems*. MIT Press, 2013, pp. 431–438.

[29] K. Imamura, J. A. Foster, and A. W. Krings. The Test Vector Problem and Limitations to Evolving Digital Circuits. In: *Proc. of the Second NASA/DoD Workshop on Evolvable Hardware*. IEEE Computer Society, 2000, pp. 75–79.

[30] J. Jeffers and J. Reinders. *Intel® Xeon Phi coprocessor high-performance programming*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013. ISBN: 978-0-12-410414-3. URL: http://opac.inria.fr/record=b1134666.

[31] K. Kannappan et al. Analyzing a Decade of Human-Competitive ("HUMIE") Winners: What Can We Learn? In: *Genetic Programming Theory and Practice XII*. Ed. by R. Riolo, W. P. Worzel, and M. Kotanchek. Cham: Springer International Publishing, 2015, pp. 149–166. ISBN: 978-3-319-16030-6. DOI: 10.1007/978-3-319-16030-6_9. URL: http://dx.doi.org/10.1007/978-3-319-16030-6_9.

[32] G. Karakonstantis and K. Roy. Voltage over-scaling: A cross-layer design perspective for energy efficient systems. In: *Circuit Theory and Design (ECCTD), 2011 20th European Conference on*. 2011, pp. 548–551. DOI: 10.1109/ECCTD.2011.6043592.

[33] P. Kaufmann and M. Platzner. Toward Self-adaptive Embedded Systems: Multi-objective Hardware Evolution. In: *Proceedings of the 20th International Conference on Architecture of Computing Systems*. ARCS'07. Zurich, Switzerland: Springer-Verlag, 2007, pp. 199–208. ISBN: 978-3-540-71267-1. URL: http://dl.acm.org/citation.cfm?id=1763274.1763289.

[34] T. Knieper et al. On Robust Evolution of Digital Hardware. In: *Biologically-Inspired Collaborative Computing: IFIP 20th World Computer Congress, Second IFIP TC 10 International Conference on Biologically-Inspired Collaborative Computing, September 8–9, 2008, Milano, Italy*. Ed. by M. Hinchey et al. Boston, MA: Springer US, 2008, pp. 213–222. ISBN: 978-0-387-09655-1. DOI: 10.1007/978-0-387-09655-1_19. URL: http://dx.doi.org/10.1007/978-0-387-09655-1_19.

[35] J. R. Koza. Human-competitive Results Produced by Genetic Programming. In: *Genetic Programming and Evolvable Machines* 11.3-4 (Sept. 2010), pp. 251–284. ISSN: 1389-2576. DOI: 10.1007/s10710-010-9112-3. URL: http://dx.doi.org/10.1007/s10710-010-9112-3.

[36] P. Krause and I. Polian. Adaptive voltage over-scaling for resilient applications. In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*. 2011, pp. 1–6. DOI: 10.1109/DATE.2011.5763153.

[37] P. Kulkarni, P. Gupta, and M. Ercegovac. Trading Accuracy for Power with an Underdesigned Multiplier Architecture. In: *VLSI Design (VLSI Design), 2011 24th International Conference on*. 2011, pp. 346–351. DOI: 10.1109/VLSID.2011.51.

[38] S.-L. Lu. Speeding up processing with approximation circuits. In: *Computer* 37.3 (2004), pp. 67–73. ISSN: 0018-9162. DOI: 10.1109/MC.2004.1274006.

[39] J. F. Miller, ed. *Cartesian Genetic Programming*. Natural Computing Series. Berlin, DE: Springer Verlag, 2011, p. 334. ISBN: 978-3-642-17309-7. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=7299.

[40] J. F. Miller and S. L. Smith. Redundancy and computational efficiency in Cartesian genetic programming. In: *Evolutionary Computation, IEEE Transactions on* 10.2 (2006), pp. 167–174. ISSN: 1089-778X. DOI: 10.1109/TEVC.2006.871253.

[41] J. F. Miller and P. Thomson. Cartesian Genetic Programming. In: *Genetic Programming*. Vol. 1802. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2000, pp. 121–132. ISBN: 978-3-540-67339-2. DOI: 10.1007/978-3-540-46239-2_9. URL: http://dx.doi.org/10.1007/978-3-540-46239-2_9.

[42] S. Mittal. A Survey of Techniques for Approximate Computing. In: *ACM Comput. Surv.* 48.4 (Mar. 2016), 62:1–62:33. ISSN: 0360-0300. DOI: 10.1145/2893356. URL: http://doi.acm.org.ezproxy.lib.vutbr.cz/10.1145/2893356.

[43] V. Mrázek and Z. Vašíček. Evolutionary Design of Transistor Level Digital Circuits using Discrete Simulation. In: *Genetic Programming, 18th European Conference, EuroGP 2015*. LCNS 9025. Berlin, DE: Springer International Publishing, 2015, pp. 66–77. ISBN: 978-3-319-16500-4. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=10774.

[44] V. Mrázek et al. EvoApprox8b: Library of Approximate Adders and Multipliers for Circuit Design and Benchmarking of Approximation Methods. In: *Proc. of the 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Lausanne, CH: European Design and Automation Association, 2017, pp. 258–261. ISBN: 978-3-9815370-9-3. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=11262.

[45] K. Nepal et al. ABACUS: A technique for automated behavioral synthesis of approximate computing circuits. In: *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*. 2014, pp. 1–6. DOI: 10.7873/DATE.2014.374.

[46] K. Nepal et al. Automated High-Level Generation of Low-Power Approximate Computing Circuits. In: *IEEE Transactions on Emerging Topics in Computing* PP.99 (2017), pp. 1–1. ISSN: 2168-6750. DOI: 10.1109/TETC.2016.2598283.

[47] J. Nevoral, R. Růžička, and V. Mrázek. Evolutionary Design of Polymorphic Gates Using Ambipolar Transistors. In: *2016 IEEE Symposium Series on Computational Intelligence*. Athens, GR: Institute of Electrical and Electronics Engineers, 2016, pp. 1–8. ISBN: 978-1-5090-4240-1. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=11239.

[48] J. Petrlik and L. Sekanina. Multiobjective evolution of approximate multiple constant multipliers. In: *IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems 2013*. Brno, CZ: IEEE Computer Society, 2013, pp. 116–119. ISBN: 978-1-4673-6133-0. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=10237.

[49] S. Picek et al. Cryptographic Boolean functions: One output, many design criteria. In: *Applied Soft Computing Journal* 40 (2016). cited By 6, pp. 635–653. DOI: 10.1016/j.asoc.2015.10.066. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84953896436&doi=10.1016%2fj.asoc.2015.10.066&partnerID=40&md5=84d66701beeec6eed0b90430ca8b71dd.

[50] S. Picek et al. Evolutionary algorithms for boolean functions in diverse domains of cryptography. In: *Evolutionary Computation* 24.4 (2016). cited By 0, pp. 667–694. DOI: 10.1162/EVCO_a_00190. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85002141900&doi=10.1162%2fEVCO_a_00190&partnerID=40&md5=63125ab5249a1482b8d841fa1ac9d190.

[51] E. Popovici et al. Coevolutionary Principles. In: *Handbook of Natural Computing*. Springer Berlin Heidelberg, 2012, pp. 987–1033. ISBN: 978-3-540-92909-3.

[52] A. Ranjan et al. ASLAN: Synthesis of approximate sequential circuits. In: *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*. 2014, pp. 1–6. DOI: 10.7873/DATE.2014.377.

[53] R. Salvador et al. Implementation Techniques for Evolvable HW Systems: Virtual vs. Dynamic Reconfiguration. In: *Proc. of the 22nd International Conference on Field Programmable Logic and Applications (FPL)*. Oslo, NO: IEEE Computer Society, 2012, pp. 547–550. ISBN: 978-1-4673-2257-7. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=9985.

[54] A. Sampson et al. EnerJ: Approximate Data Types for Safe and General Low-power Computation. In: *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '11. San Jose, California, USA: ACM, 2011, pp. 164–174. ISBN: 978-1-4503-0663-8. DOI: 10.1145/1993498.1993518. URL: http://doi.acm.org/10.1145/1993498.1993518.

[55] L. Sekanina. *Evolvable Components - From Theory to Hardware Implementations*. Natural Computing Series. Berlin, DE: Springer Verlag, 2003, p. 194. ISBN: 3-540-40377-9. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=7299.

[56] L. Sekanina. Evolvable hardware. In: *Handbook of Natural Computing*. Berlin, DE: Springer Verlag, 2012, pp. 1657–1705. ISBN: 978-3-540-92909-3. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=10107.

[57] L. Sekanina. Image Filter Design with Evolvable Hardware. In: *Lecture Notes in Computer Science* 2002.2279 (2002), pp. 255–266. ISSN: 0302-9743. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=6889.

[58] L. Sekanina and Z. Vasicek. Approximate Circuits by Means of Evolvable Hardware. In: *2013 IEEE International Conference on Evolvable Systems (ICES)*. Proceedings of the 2013 IEEE Symposium Series on Computational Intelligence (SSCI). Singapur, SG: IEEE Computer Society, 2013, pp. 21–28. ISBN: 978-1-4673-5847-7. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=10198.

[59] L. Sekanina and Z. Vasicek. Evolutionary Computing in Approximate Circuit Design and Optimization. In: *1st Workshop on Approximate Computing (WAPCO 2015)*. Amsterdam, 2015, pp. 1–6. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=10765.

[60] M. Tomassini. *Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time (Natural Computing Series)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005. ISBN: 3540241930.

[61] G. Toscano Pulido and C. A. Coello Coello. The Micro Genetic Algorithm 2: Towards Online Adaptation in Evolutionary Multiobjective Optimization. In: *Evolutionary Multi-Criterion Optimization: Second International Conference, EMO 2003, Faro, Portugal, April 8–11, 2003. Proceedings*. Ed. by C. M. Fonseca et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 252–266. ISBN: 978-3-540-36970-7. DOI: 10.1007/3-540-36970-8_18. URL: http://dx.doi.org/10.1007/3-540-36970-8_18.

[62] M. Trefzer and A. Tyrrell. *Evolvable Hardware: From Practice to Application*. Natural Computing Series. Springer Berlin Heidelberg, 2015. ISBN: 9783662446164. URL: https://books.google.cz/books?id=-Y2QCgAAQBAJ.

[63] M. Trefzer et al. Operational Amplifiers: An Example for Multi-objective Optimization on an Analog Evolvable Hardware Platform. In: *Proceedings of the 6th International Conference on Evolvable Systems: From Biology to Hardware*. ICES'05. Sitges, Spain: Springer-Verlag, 2005, pp. 86–97. ISBN: 3-540-28736-1, 978-3-540-28736-0. DOI: 10.1007/11549703_9. URL: http://dx.doi.org/10.1007/11549703_9.

[64] S. Tsutsui and P. Collet. *Massively Parallel Evolutionary Computation on GPGPUs*. Natural Computing Series. Springer Berlin Heidelberg, 2013. ISBN: 9783642379581. URL: http://books.google.cz/books?id=ucnAmgEACAAJ.

[65] A. J. Turner and J. F. Miller. Recurrent Cartesian Genetic Programming. English. In: *Parallel Problem Solving from Nature – PPSN XIII*. Vol. 8672. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 476–486. ISBN: 978-3-319-10761-5. DOI: 10.1007/978-3-319-10762-2_47. URL: http://dx.doi.org/10.1007/978-3-319-10762-2_47.

[66] L. Vanneschi and R. Poli. Genetic Programming – Introduction, Applications, Theory and Open Issues. In: *Handbook of Natural Computing*. Springer Berlin Heidelberg, 2012, pp. 709–739. ISBN: 978-3-540-92909-3.

[67] Z. Vasicek and L. Sekanina. Evolutionary Approach to Approximate Digital Circuits Design. In: *IEEE Transactions on Evolutionary Computation* 99.99 (2015), pp. 1–13. ISSN: 1089-778X. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=10406.

[68] Z. Vasicek and L. Sekanina. Evolutionary Design of Approximate Multipliers Under Different Error Metrics. In: *17th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*. Warsaw, PL: IEEE Computer Society, 2014, pp. 135–140. ISBN: 978-1-4799-4558-0. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=10513.

[69] Z. Vasicek and L. Sekanina. Formal Verification of Candidate Solutions for Post-Synthesis Evolutionary Optimization in Evolvable Hardware. In: *Genetic Programming and Evolvable Machines* 12.3 (2011), pp. 305–327.

[70] Z. Vasicek and L. Sekanina. Hardware Accelerator of Cartesian Genetic Programming with Multiple Fitness Units. In: *Computing and Informatics* 29.6 (2010), pp. 1359–1371. ISSN: 1335-9150. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=9421.

[71] Z. Vasicek and L. Sekanina. How to Evolve Complex Combinational Circuits From Scratch? In: *2014 IEEE International Conference on Evolvable Systems Proceedings*. Piscataway, US: Institute of Electrical and Electronics Engineers, 2014, pp. 133–140. ISBN: 978-1-4799-4480-4. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=10673.

[72] Z. Vasicek and L. Sekanina. On Area Minimization of Complex Combinational Circuits Using Cartesian Genetic Programming. In: *2012 IEEE World Congress on Computational Intelligence*. CA, US: Institute of Electrical and Electronics Engineers, 2012, pp. 2379–2386. ISBN: 978-1-4673-1508-1. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=9866.

[73] Z. Vasicek and K. Slany. Efficient Phenotype Evaluation in Cartesian Genetic Programming. In: *Proc. of the 15th European Conference on Genetic Programming*. LNCS 7244. Springer Verlag, 2012, pp. 266–278.

[74] F. Vaverka, R. Hrbáček, and L. Sekanina. Evolving Component Library for Approximate High Level Synthesis. In: *2016 IEEE Symposium Series on Computational Intelligence*. Athens, GR: IEEE Computational Intelligence Society, 2016, pp. 1–8. ISBN: 978-1-5090-4240-1. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=11231.

[75] S. Venkataramani et al. SALSA: Systematic logic synthesis of approximate circuits. In: *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*. 2012, pp. 796–801.

[76] S. Venkataramani, K. Roy, and A. Raghunathan. Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*. 2013, pp. 1367–1372. DOI: 10.7873/DATE.2013.280.

[77] R. Venkatesan et al. MACACO: Modeling and analysis of circuits for approximate computing. In: *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*. 2011, pp. 667–673. DOI: 10.1109/ICCAD.2011.6105401.

[78] J. A. Walker and J. F. Miller. Embedded Cartesian Genetic Programming and the Lawnmower and Hierarchical-if-and-only-if Problems. In: *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*. GECCO '06. Seattle, Washington, USA: ACM, 2006, pp. 911–918. ISBN: 1-59593-186-4. DOI: 10.1145/1143997.1144154. URL: http://doi.acm.org/10.1145/1143997.1144154.

[79] J. A. Walker, J. F. Miller, and R. Cavill. A Multi-chromosome Approach to Standard and Embedded Cartesian Genetic Programming. In: *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*. GECCO '06. Seattle, Washington, USA: ACM, 2006, pp. 903–910. ISBN: 1-59593-186-4. DOI: 10.1145/1143997.1144153. URL: http://doi.acm.org/10.1145/1143997.1144153.

[80] J. A. Walker et al. Parallel evolution using multi-chromosome cartesian genetic programming. English. In: *Genetic Programming and Evolvable Machines* 10.4 (2009), pp. 417–445. ISSN: 1389-2576. DOI: 10.1007/s10710-009-9093-2. URL: http://dx.doi.org/10.1007/s10710-009-9093-2.

[81] M. Wiglasz and M. Drahošová. Plastic Fitness Predictors Coevolved with Cartesian Programs. In: *19th European Conference on Genetic programming*. LNCS 9594. Berlin, DE: Springer International Publishing, 2016, pp. 164–179. ISBN: 978-3-319-30667-4. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=11001.

[82] H. Yu, Y. Ha, and J. Wang. Quality Optimization of Resilient Applications Under Temperature Constraints. In: *Proceedings of the Computing Frontiers Conference*. CF'17. Siena, Italy: ACM, 2017, pp. 9–16. ISBN: 978-1-4503-4487-6. DOI: 10.1145/3075564.3075577. URL: http://doi.acm.org/10.1145/3075564.3075577.

[83] T. Yu and J. Miller. Finding Needles in Haystacks Is Not Hard with Neutrality. English. In: *Genetic Programming*. Vol. 2278. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, pp. 13–25. ISBN: 978-3-540-43378-1. DOI: 10.1007/3-540-45984-7_2. URL: http://dx.doi.org/10.1007/3-540-45984-7_2.

[84] E. Zitzler, M. Laumanns, and L. Thiele. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*. Tech. rep. 2001.

# Appendix A

# Related Publications

## A.1 Towards Highly Optimized Cartesian Genetic Programming: From Sequential via SIMD and Thread to Massive Parallel Implementation

Radek Hrbacek and Lukas Sekanina

In *GECCO '14 Proceedings of the 2014 conference on Genetic and evolutionary computation.*
GECCO 2014, New York, US, Association for Computing Machinery, 2014.
ISBN: 978-1-4503-2662-9, pp. 1015-1022.

# Towards Highly Optimized Cartesian Genetic Programming: From Sequential via SIMD and Thread to Massive Parallel Implementation

Radek Hrbacek
Brno University of Technology, Czech republic
Faculty of Information Technology
ihrbacek@fit.vutbr.cz

Lukas Sekanina
Brno University of Technology, Czech republic
Faculty of Information Technology
sekanina@fit.vutbr.cz

## ABSTRACT

Most implementations of Cartesian genetic programming (CGP) which can be found in the literature are sequential. However, solving complex design problems by means of genetic programming requires parallel implementations of search methods and fitness functions. This paper deals with the design of highly optimized implementations of CGP and their detailed evaluation in the task of evolutionary circuit design. Several sequential implementations of CGP have been analyzed and the effect of various additional optimizations has been investigated. Furthermore, the parallelism at the instruction, data, thread and process level has been applied in order to take advantage of modern processor architectures and computer clusters. Combinational adders and multipliers have been chosen to give a performance comparison with state of the art methods.

## Categories and Subject Descriptors

B.6.0 [**Hardware**]: Logic Design—*General*; I.2.8 [**Computing methodologies**]: Artificial intelligence—*Problem Solving, Control Methods, and Search*

## Keywords

Cartesian Genetic Programming, Parallel Computing, SIMD, AVX, Cluster, Combinational Circuit Design

## 1. INTRODUCTION

The evolutionary design conducted by means of genetic programming (GP) is a very computationally demanding design method. In order to reduce the design time, various accelerators of genetic programming have been proposed. The accelerators are typically developed to speed up the main components of the method – the search algorithm and the fitness evaluation procedure. While the former case is usually investigated in the field of parallel and distributed evolutionary algorithms, the latter one typically involves accelerating an application specific simulator in which every candidate phenotype has to be evaluated. The most famous parallel approaches to GP are represented by Koza's Beowulf-style parallel cluster [10] and recently, an approach developed for the cloud environment [17].

As CGP has been accelerated on GPUs [6] and FPGAs [20], in this contribution, the parallelism at the instruction, data, thread and process level has been applied in order to take advantage of modern processor architectures and computer clusters. The goal of this paper is to provide a set of parallel CGP implementations that can be used on these widely accessible parallel computers. The proposed implementations will be compared and evaluated in the task of adder and multiplier evolutionary design. The reason for choosing these two problems is that the literature includes several case studies that can be used for comparative purposes.

Another important assumption of this study is that CGP starts with a randomly generated initial population and the objective is to find a fully functional solution, i.e. an $n_{\mathrm{i}}$-input/$n_{\mathrm{o}}$-output circuit which provides $n_{\mathrm{o}} \cdot 2^{n_{\mathrm{i}}}$ correct output bits when all possible input combinations are evaluated. In other words, the goal is not in minimizing the number of gates, delay or other criteria. Note that for the evolutionary circuit optimization, in which CGP starts with a fully functional solution and the goal is to minimize the number of gates, efficient and fast fitness calculation methods based on formal functional equivalence checking algorithms have already been proposed [21]. While such methods are capable of optimizing circuits having hundreds of inputs and thousands of gates, only relatively small circuits (with ten to fifteen inputs and less than 100 gates) have been evolved so far in the proposed scenario. Finally, this work does not take into account advanced methods such as divide and conquer [18, 16] or self-modifying CGP [7] which allow for reducing the problem complexity and consequently applying the standard CGP on sub-problems. Therefore, all techniques reported in this paper operate on the whole circuit.

Parallel CGP implementations are usually focused on an efficient phenotype evaluation, which is the most time critical operation of CGP due to the fact that the circuit evaluation time grows exponentially with the number of circuit inputs. In order to accelerate a candidate circuit evaluation, one can apply a parallel evaluation of multiple training vectors by means of bit-level instructions [14], circuit precompilation techniques or streaming SIMD extensions (SSE) of modern processors [22].


*GECCO'14,* July 12-16, 2014, Vancouver, BC, Canada.

http://dx.doi.org/10.1145/2576768.2598343

On the other hand, the search algorithm used in the standard CGP is a simple (1+$\lambda$) evolution strategy. A natural approach to accelerating the search is evaluating $\lambda$ offspring on $\lambda$ processors in parallel. A few attempts were made to introduce more advanced operators into this search method, but only a small improvement was reported in [3]. However, a noticeable improvement can be obtained when the standard CGP is replaced by parallel coevolutionary CGP [9].

The rest of the paper is organized as follows. Section 2 introduces CGP and its usage as combinational circuit design method. The implementation of several sequential solutions is discussed in Section 3. Section 4 deals with the CGP parallelisation. Section 5 is dedicated to experiments and the achieved results. Final conclusions can be found in Section 6.

## 2. CARTESIAN GENETIC PROGRAMMING

Cartesian genetic programming has been introduced by Miller [12] as a branch of genetic programming. Unlike GP which uses tree representation, an individual in CGP is represented by a directed acyclic graph which enables the candidate solution to have multiple outputs and automatically reuse intermediate results. This makes CGP very suitable for design of various kinds of digital circuits, such as arithmetic and logic circuits, digital filters, etc. [13].

CGP uses a cartesian grid of $n_\mathrm{r} \times n_\mathrm{c}$ programmable elements (nodes) interconnected by a feed-forward network (Figure 1). Each node's input (usually each node has a fixed number of inputs $n_\mathrm{ni} = 2$) can be connected either to one of $n_\mathrm{i}$ primary inputs or to a node output in the preceding $l$ columns. By setting the $l$-back parameter and the grid size, one can control the area and delay of the circuit. Each node can be programmed to perform one of $n_\mathrm{ni}$-input functions defined in the set $\Gamma$ (let $n_\mathrm{f} = |\Gamma|$). The $n_\mathrm{o}$ primary circuit outputs are connected either to the primary inputs or nodes. The output connectivity can be optionally restricted by the $o$-back parameter.

Since all the CGP parameters are fixed, each chromosome is encoded using a fixed-sized array of $n_\mathrm{r} \cdot n_\mathrm{c} \cdot (n_\mathrm{ni} + 1) + n_\mathrm{o}$ integers. Each primary input is assigned a number from $\{0, ..., n_\mathrm{i} - 1\}$ and the nodes are assigned numbers from $\{n_\mathrm{i}, ..., n_\mathrm{i} + n_\mathrm{r} \cdot n_\mathrm{c} - 1\}$. The genotype is of fixed length, whereas the phenotype is of variable length depending on the number of inactive nodes, i.e. nodes whose output is not used by any other node or primary output. Hence, the genotype-phenotype mapping is not injective. The existence of genotypes with the same fitness is usually referred to as neutrality. The role of neutrality has been intensively

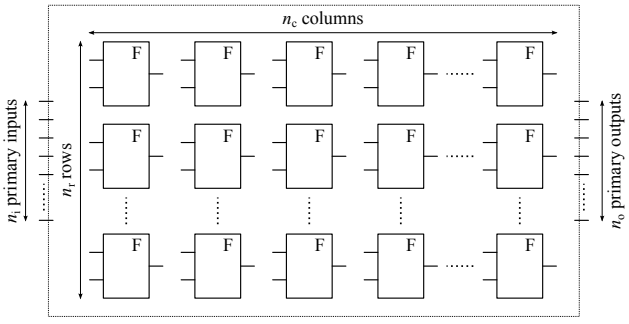



**Figure 1: Cartesian genetic programming scheme.**

studied [24] and it was shown that for certain problems the neutrality significantly reduces the computational effort and helps to find more innovative solutions [11].

In CGP, a simple mutation based $(1 + \lambda)$ evolutionary strategy is used as a search mechanism. The population size $1 + \lambda$ is usually very small, typically, $\lambda$ is between 1 and 15. The initial population is constructed either randomly (evolutionary design) or by mapping of a known solution to the CGP chromosome (evolutionary optimization) [21]. In each generation, a randomly selected individual with the best fitness value is passed to the next generation unmodified and its $\lambda$ offspring individuals are created by means of point mutation operator which modifies $m$ randomly selected genes of the chromosome. The mutation rate $m$ is usually set to modify up to 5 % of the genes. Despite numerous attempts, no useful crossover operator has been introduced. For some problem classes (e.g. symbolic regression problem), special crossover operators have been investigated [3], however, in the case of digital circuits design, none of them has been confirmed as useful.

In the case of combinational circuit evolution, the fitness function corresponds to the quality of the candidate circuit measured as the number of correct output bits compared to a specified truth table. In order to obtain a fully working circuit, all combinations of input values have to be evaluated. For a circuit with $n_\mathrm{i}$ inputs and $n_\mathrm{o}$ outputs, $2^{n_\mathrm{i}}$ test vectors need to be fetched to the primary inputs and $n_\mathrm{o} \cdot 2^{n_\mathrm{i}}$ output bits have to be verified so as to compute the fitness value.

## 3. SEQUENTIAL IMPLEMENTATION

Every CGP implementation must process all the $2^{n_\mathrm{i}}$ test vectors on the whole phenotype for the entire population of individuals and compare all the $n_\mathrm{o}$ outputs to the desired ones. This requirement directly implies the presence of 3 independent nested loops – the test vector loop, the loop over all nodes and the population loop. The order of these 3 loops is crucial for the performance and the optimal choice varies among different implementations. In a very naive implementation, one can process each test vector separately on all nodes no matter if they are active or not. However, in order to take advantage of modern superscalar out-of-order processors, the parallelism at various levels has to be employed and special attention to memory access policy has to be paid.

The most fundamental optimization we can apply is the bit-level parallelism. Instead of separate test vector processing, up to 64 test vectors can be processed in parallel on 64-bit processors thanks to bitwise operations. Furthermore, by introducing the data-level parallelism using SIMD instructions, 128 or even 256 test vectors can fit into the SSE or AVX registers respectively.

One of the most commonly used optimization in CGP is the detection of inactive nodes. Before processing each individual, the genotype is traversed in the reversed order and the nodes whose output is never used are marked as inactive. While processing, all inactive nodes are skipped and thus only the phenotype is treated.

### 3.1 Interpreted implementation

The *interpreted* implementation is very simple, yet for smaller circuits very efficient. The principle is shown in Algorithm 1. At the beginning, an initial population is created randomly just like in any other implementation. Then, in

```
randomly create and evaluate initial population;
while termination condition is false do
    for i in 1 to P do
        if i = best_ind then
            continue;
        end
        copy ind[best_ind] chromosome to ind[i];
        mutate ind[i];
        analyze ind[i];
        foreach node do
            foreach test vector do
                compute node output value;
            end
        end
        foreach primary output do
            ind[i].fit += number of wrong bits;
        end
        if ind[i].fit > ind[best_ind].fit then
            best_ind := i;
        end
    end
end
```

**Algorithm 1:** Interpreted implementation

every generation, the evaluation of the population is going on as follows: For each individual not being the best individual from the previous generation, the chromosome of the best individual is copied and mutated. After that, the chromosome is analyzed in order to find the inactive nodes. For each active node, all test vectors are processed according to the node function. The test vector loop is put inside the node loop because of the overhead of the `switch` statement the node function is based on. Besides the overhead, by putting the test vector loop inside, the compiler is able to optimize this loop by unrolling. After computing each node's output value, the primary outputs are checked against desired values and the number of wrong bits is accumulated. This can be done very efficiently just by `XOR`ing the actual output value and the expected value and counting the number of ones. Since SSE 4.2, a special instruction `POPCNT` exists which allows to count the number of ones with the latency of 3 clock cycles (on the Intel Sandy Bridge microarchitecture) [4].

Since all of the intermediate results for all test vectors have to be kept in memory during the evaluation, the memory usage of the interpreted implementation is not optimal (up to $n_r \cdot n_c \cdot 2^{n_i}/8$ bytes), but it is still efficient for small circuits, until the required memory size exceeds the cache size.

## 3.2 Native implementation

By introducing the *native* implementation, both the memory requirements and the `switch` statement overhead can be significantly reduced. The principle can be seen from Algorithm 2. Just like in the interpreted implementation, each individual except the previous best one is copied, mutated and analyzed. The difference lies in the evaluation process

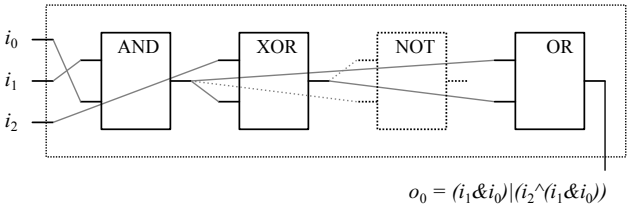



**Figure 2: CGP individual example.**

```
randomly create and evaluate initial population;
while termination condition is false do
    for i in 1 to P do
        if i = best_ind then
            continue;
        end
        copy ind[best_ind] chromosome to ind[i];
        mutate ind[i];
        analyze ind[i];
        compile ind[i];
    end
    foreach test vector do
        for i in 1 to P do
            if i = best_ind then
                continue;
            end
            ind[i].fit += run compiled code;
        end
    end
    for i in 1 to P do
        if ind[i].fit > ind[best_ind].fit then
            best_ind := i;
        end
    end
end
```

**Algorithm 2:** Native implementation.

– instead of traversing the chromosome and computing the node outputs directly, the chromosome is compiled at first. The compiled program is then executed on each test vector for each individual in the population. This technique was first introduced in [22] and the implementation presented in this paper further improves this principle by introducing subroutine parameters, native fitness calculation and by utilizing AVX instructions. Moreover, the native implementation uses slightly less memory since there is no need to keep the intermediate results for all test vectors.

Figure 2 depicts a very simple circuit with $n_i = 3$ primary inputs, $n_o = 1$ primary output and $n_c = 4$ nodes. Listing 1 shows the compiled chromosome from Figure 2 in 64-bit version. The compiled subroutine has 4 parameters passed on in the 64-bit registers `RDI`, `RSI`, `RDX` and `RCX` [15] – the pointers to the primary inputs filled by corresponding test vectors, node outputs, desired primary outputs and spe-

**Listing 1: Native 64b implementation example.**

```
push    %rbx            ; store RBX

mov     0x08(%rsi),%rbx ; node n0
and     0x00(%rsi),%rbx ; AND
mov     %rbx,0x18(%rdi) ; n0 := i1 AND i0

mov     0x10(%rsi),%rbx ; node n1
xor     0x18(%rdi),%rbx ; XOR
mov     %rbx,0x20(%rdi) ; n1 := i2 XOR n0

mov     0x18(%rdi),%rbx ; node n3
or      0x28(%rdi),%rbx ; OR
mov     %rbx,0x30(%rdi) ; n3 := n0 OR n2

xor     %rax,%rax       ; RAX := 0

mov     0x30(%rsi),%rbx ; output 00
xor     0x0(%rdx),%rbx
and     0x0(%rcx),%rbx  ; mask m0
popcnt  %rbx,%bdx       ; error count
add     %rbx,%rax       ; accumulate

pop     %rbx            ; restore RBX
retq                    ; return RAX
```

cial masks.[1] For each node, the first input value is loaded from the memory to the RBX register and the desired operation is performed (optionally, the second input value is loaded).[2] The node output is then stored back to the memory. After processing all active nodes, the number of wrong output bits is accumulated in the RAX register. For each primary output, the corresponding node output is loaded from the memory and XORed with the desired value. After applying the mask, the number of incorrect bits is computed using the POPCNT instruction and accumulated in the RAX register. The register RAX is used for integer return values [15], thus the subroutine returns the number of wrong bits for a given test vector.

The same example compiled in the AVX version can be seen in Listing 2. Here, the calling convention is the same and the register RAX has the same purpose as in the 64-bit version. The intermediate results are computed in the YMM0 register. The register YMM1 contains just ones and serves for computing the NOT operation using XOR, since there is not an AVX instruction for this purpose. Compared to the 64-bit version, the error computation is more complicated as there is only a 32-bit and 64-bit POPCNT instruction available.

[1]After XORing the actual and desired output value, the result is ANDed with this mask, which enables to specify which output bits are not considered (we don't care about their values).

[2]There is no need to avoid the output dependency thanks to hardware register renaming.

**Listing 2: Native AVX implementation example.**

```
push    %rbx                    ; store RBX

mov     0xXXXXXXXX,%rax         ; RAX := &avx_ones
vmovdqa 0x0(%rax),%ymm1         ; YMM1 := 111..111

vmovdqa 0x20(%rsi),%ymm0        ; node n0
vandps  0x0(%rdi),%ymm0,%ymm0   ; AND
vmovdqa %ymm0,0x60(%rdi)        ; n0 := i1 AND i0

vmovdqa 0x40(%rsi),%ymm0        ; node n1
vxorps  0x60(%rdi),%ymm0,%ymm0  ; XOR
vmovdqa %ymm0,0x80(%rdi)        ; n1 := i2 XOR n0

vmovdqa 0x60(%rdi),%ymm0        ; node n3
vorps   0xa0(%rdi),%ymm0        ; OR
vmovdqa %ymm0,0xc0(%rdi)        ; n3 := n0 OR n2

xor     %rax,%rax               ; RAX := 0

mov     0xc0(%rsi),%rbx         ; output o0[0]
xor     0x0(%rdx),%rbx
and     0x0(%rcx),%rbx          ; mask m0[0]
popcnt  %rbx,%rbx               ; error count
add     %rbx,%rax
mov     0xc8(%rsi),%rbx         ; output o0[1]
xor     0x8(%rdx),%rbx
and     0x8(%rcx),%rbx          ; mask m0[1]
popcnt  %rbx,%rbx               ; error count
add     %rbx,%rax
mov     0xd0(%rsi),%rbx         ; output o0[2]
xor     0x10(%rdx),%rbx
and     0x10(%rcx),%rbx         ; mask m0[2]
popcnt  %rbx,%rbx               ; error count
add     %rbx,%rax
mov     0xd8(%rsi),%rbx         ; output o0[3]
xor     0x18(%rdx),%rbx
and     0x18(%rcx),%rbx         ; mask m0[3]
popcnt  %rbx,%rbx               ; error count
add     %rbx,%rax

pop     %rbx                    ; restore RBX
retq                            ; return RAX
```

The native implementation enables to introduce more optimizations than the interpreted implementation. The fitness computation can be stopped after exceeding the number of wrong bits matching the best individual. Both native and interpreted implementations can detect neutral mutations and skip recomputing fitness values for individuals affected only by neutral mutations [5].

The efficiency of the native implementation lies in exploiting the instruction-level parallelism offered by modern superscalar out-of-order processors by reducing branch mispredictions and cache misses and increasing the arithmetic intesity.

# 4. PARALLEL IMPLEMENTATION

Until the beginning of the 21st century, the aim of the processor architects was to increase the single threaded performance by means of extracting more instruction-level parallelism (ILP) and utilizing superscalar out-of-order execution, sophisticated branch predictors, multi-level cache hierarchy, etc. However, growing power consumption and limited ILP extractable from common sequential code together with increasing transistor density led to the introduction of multiprocessors [8]. Since then, special attention has to be paid to parallel computing in order to make use of modern processor architectures.

## 4.1 Thread parallelism

The purpose of the thread-level parallelism (TLP) in CGP is to speed up the whole evolutionary process – both the fitness calculation and the genetic operators. Both interpreted and native parallel implementations are discussed in this subsection; the OpenMP library has been used for managing the threads.

Algorithm 3 shows the scheme of the interpreted parallel implementation, very similar to the corresponding sequential variant. The outer population loop has to be scheduled dynamically, because if it were scheduled statically, the thread responsible for the previous best individual would have less work than the others resulting in poor load balancing.

The parallelization of the native implementation is some-

```
randomly create and evaluate initial population;
while termination condition is false do
    #pragma omp for schedule(dynamic)
    for i in 1 to pop_size do
        if i = best_ind then
            continue;
        end
        copy ind[best_ind] chromosome to ind[i];
        mutate ind[i];
        analyze ind[i];
        foreach node do
            foreach test vector do
                compute node output value;
            end
        end
        foreach primary output do
            ind[i].fit += number of wrong bits;
        end
        #pragma omp critical
        if ind[i].fit > ind[best_ind].fit then
            best_ind := i;
        end
    end
end
```

**Algorithm 3:** Parallel interpreted implementation.


**Figure 3: Parallel native implementation threading.**

```
randomly create and evaluate initial population;
while termination condition is false do
    #pragma omp for
    for i in 1 to P do
        if i = best_ind then
            continue;
        end
        copy ind[best_ind] chromosome to ind[i];
        mutate ind[i];
        analyze ind[i];
        compile ind[i];
    end
    foreach i in 1 to P do
        fit[i] = 0;
    end
    #pragma omp barrier
    #pragma omp for nowait
    foreach test vector do
        foreach i = 1 to P do
            if i = best_ind then
                continue;
            end
            fit[i] += run compiled code;
        end
    end
    foreach i in 1 to P do
        #pragma omp atomic
        ind[i].fit += fit[i];
    end
    #pragma omp barrier
    #pragma omp single
    for i in 1 to P do
        if ind[i].fit > ind[best_ind].fit then
            best_ind := i;
        end
    end
end
```

**Algorithm 4:** Parallel native implementation.

what more complicated, since it consists of three separate loops, each having a different number of iterations. Figure 3 depicts the threading scheme. At first, a new population has to be created using up to $P$ threads, then the evaluation can utilize all the threads (if there are enough test vectors) and at the end, the new best individual has to be found, unfortunately by a single thread.

Algorithm 4 shows the overall parallel implementation principle. A special attention had to be paid for the fitness accumulation across test vectors treated by different threads. Each thread has its own partial fitness values which are finally atomically accumulated to the total fitness values doing a manual reduction over the fitness values (OpenMP offers only reduction over scalar variables).

## 4.2 Process parallelism (inter-population)

Spatially structured evolutionary algorithms have been intensively studied in the past and a variety of approaches differing in the used evolutionary algorithm or communication topology has emerged [19, 2]. By introducing multiple populations evolving in parallel, one can increase the population

```
seed := randomly generated individual;
while termination condition is false do
    run evolutionary design starting with seed;
    exchange best individuals among islands;
    if global best fitness is higher than local then
        seed := global best individual;
    end
end
```

**Algorithm 5:** Isolated islands model.

diversity and thus make the EA more explorative leading to a higher probability of finding the global optimum for particular problems.

The combinational circuit design is a very complex problem, the search space is generally rugged containing lots of local optima and thus the potential of exploiting parallel EA is high. Unfortunately, the absence of a crossover operator in CGP is a very limiting factor since most parallel models take advantage of combining genotypes from different isolated populations. Nevertheless, the model of isolated islands with migration of the best individuals in each population can be applied to this problem.

Algorithm 5 describes the parallel evolutionary process. At the beginning, each population starts with a randomly generated initial population. Until a perfectly working circuit is found, the evolutionary process is executed on each island and after specified number of generations, the best individual from each island is broadcasted to the other islands. If the global best individual has higher fitness value than the local best individual, the island is seeded by the global best one.

After migration, each isolated population is evolving independently and can explore different areas in the search space. This makes the search algorithm more effective and speeds up the evolutionary process.

The implementation is based on the Open MPI library and can be executed on computer clusters of arbitrary size as well as on a single multiprocessor giving a great scalability to the evolutionary design process.

# 5. EXPERIMENTAL RESULTS

In this section, experiments regarding the implementation performance are presented and the scalability of the implementation is demonstrated on selected combinational circuit design problems. All experiments were performed on a computer cluster of 112 nodes with the following hardware configuration: 2× 8-core Intel E5-2670, 128 GB RAM, 2× 600 GB 15 k scratch hard disks, connected by gigabit Ethernet and Infiniband links.

The implementations have been examined by means of the common metrics: *speedup*, defined as the ratio of the sequential implementation execution time to the parallel execution time, and *efficiency*, the ratio between the achieved speedup and the number of threads.

## 5.1 Sequential implementation efficiency

The performance of the sequential implementations has been measured in the task of a combinational adder design. Table 1 and Figure 4 summarize the mean evolution times obtained from 100 independent runs for individual sequential implementations. The CGP parameters were set as follows: population of 5 individuals, $n_c = 100$ nodes, mutation rate 5 %, $\Gamma = \{\texttt{BUF}, \texttt{NOT}, \texttt{AND}, \texttt{OR}, \texttt{XOR}, \texttt{NAND}, \texttt{NOR}, \texttt{XNOR}\}$. The goal was not to find a fully functional solution, the evolu-

**Table 1: Sequential implementation performance (combinational adders, 10 000 generations).**

| width | evolution time [s] | | | |
|---|---|---|---|---|
| | interpreted | | native | |
| | 64b | 256b | 64b | 256b |
| $1 \times 1$ | 0.00382 | - | 0.00477 | - |
| $2 \times 2$ | 0.00908 | - | 0.02168 | - |
| $3 \times 3$ | 0.01994 | - | 0.04954 | - |
| $4 \times 4$ | 0.02442 | 0.02398 | 0.05497 | 0.07092 |
| $5 \times 5$ | 0.04681 | 0.03215 | 0.07076 | 0.08550 |
| $6 \times 6$ | 0.11488 | 0.08280 | 0.11168 | 0.11109 |
| $7 \times 7$ | 0.97894 | 0.40800 | 0.28149 | 0.21899 |
| $8 \times 8$ | 6.27716 | 2.14536 | 0.88332 | 0.55520 |
| $9 \times 9$ | 32.64657 | 9.84838 | 3.63436 | 2.21870 |
| $10 \times 10$ | 154.38932 | 47.59685 | 14.99801 | 8.75244 |

**Table 2: Sequential implementation speedup (combinational adders, 10 000 generations).**

| width | speedup [-] | | |
|---|---|---|---|
| | interpreted | native | |
| | 256b | 64b | 256b |
| $1 \times 1$ | - | 0.80084 | - |
| $2 \times 2$ | - | 0.41882 | - |
| $3 \times 3$ | - | 0.40250 | - |
| $4 \times 4$ | 1.01835 | 0.44424 | 0.34433 |
| $5 \times 5$ | 1.45599 | 0.66153 | 0.54749 |
| $6 \times 6$ | 1.38744 | 1.02865 | 1.03412 |
| $7 \times 7$ | 2.39936 | 3.47771 | 4.47025 |
| $8 \times 8$ | 2.92592 | 7.10633 | 11.30612 |
| $9 \times 9$ | 3.31492 | 8.98276 | 14.71428 |
| $10 \times 10$ | 3.24369 | 10.29399 | 17.63957 |

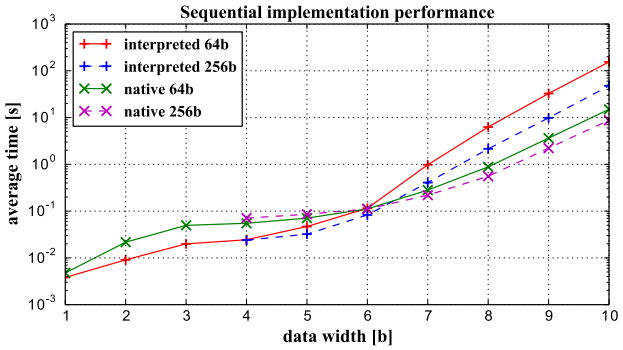



**Figure 4: Sequential implementation performance (combinational adders, 10 000 generations).**

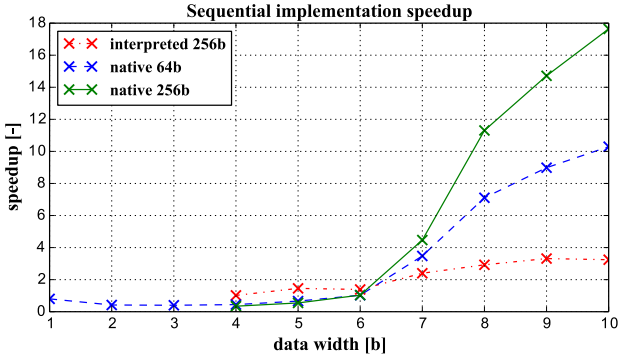



**Figure 5: Sequential implementation speedup (combinational adders, 10 000 generations).**

tion has been stopped after reaching 10 000 generations. The achieved speedup relative to the 64b interpreted implementation can be seen in Table 2 and Figure 5. The experimental results indicate that for small circuits, the best implementation is the 64b interpreted implementation. Starting with 8 primary inputs, the number of test vectors goes over the limit of 256 test vectors and the AVX implementation comes to the foreground. The native AVX implementation needs even larger circuits to overcome the compilation overhead and to be sufficiently efficient, however, the achieved speedup is significant.

## 5.2 Parallel implementation efficiency

The sequential implementation performance is not substantially dependent on the number of nodes, which is not the case of the parallel implementation efficiency. Therefore, in order to evaluate the parallel speedup and efficiency, a more complex circuit has been chosen for the experiment, namely the combinational multiplier with $n_c = 800$ nodes. The population size was 5 individuals, the evolutionary process was stopped after 100 000 generations in the case of data widths 4–6 bits, 10 000 otherwise. Table 3 summarizes

**Table 3: Parallel implementation efficiency.**

| width | threads | time [s] | speedup [-] | efficiency [%] |
|---|---|---|---|---|
| $4 \times 4$ | 1 | 1.092 | - | - |
| | 2 | 0.705 | 1.550 | 77.484 |
| | 3 | 0.683 | 1.598 | 53.277 |
| | 4 | 0.490 | 2.227 | 55.677 |
| $5 \times 5$ | 1 | 1.409 | - | - |
| | 2 | 0.857 | 1.644 | 82.213 |
| | 3 | 0.837 | 1.683 | 56.094 |
| | 4 | 0.567 | 2.486 | 62.149 |
| $6 \times 6$ | 1 | 3.616 | - | - |
| | 2 | 2.048 | 1.766 | 88.295 |
| | 3 | 1.950 | 1.855 | 61.827 |
| | 4 | 1.226 | 2.950 | 73.751 |
| $7 \times 7$ | 1 | 0.584 | - | - |
| | 2 | 0.372 | 1.571 | 78.554 |
| | 3 | 0.295 | 1.983 | 66.103 |
| | 4 | 0.247 | 2.361 | 59.033 |
| | 5 | 0.212 | 2.755 | 55.101 |
| | 6 | 0.198 | 2.947 | 49.119 |
| | 7 | 0.202 | 2.899 | 41.410 |
| | 8 | 0.196 | 2.985 | 37.312 |
| $8 \times 8$ | 1 | 1.663 | - | - |
| | 2 | 0.925 | 1.797 | 89.872 |
| | 3 | 0.678 | 2.452 | 81.745 |
| | 4 | 0.539 | 3.082 | 77.044 |
| | 5 | 0.448 | 3.710 | 74.202 |
| | 6 | 0.399 | 4.163 | 69.385 |
| | 7 | 0.380 | 4.370 | 62.424 |
| | 8 | 0.347 | 4.786 | 59.831 |
| $9 \times 9$ | 1 | 6.108 | - | - |
| | 2 | 3.251 | 1.879 | 93.945 |
| | 3 | 2.202 | 2.774 | 92.466 |
| | 4 | 1.717 | 3.556 | 88.908 |
| | 5 | 1.422 | 4.294 | 85.879 |
| | 6 | 1.230 | 4.964 | 82.730 |
| | 7 | 1.105 | 5.529 | 78.990 |
| | 8 | 0.985 | 6.199 | 77.482 |
| $10 \times 10$ | 1 | 25.825 | - | - |
| | 2 | 13.455 | 1.919 | 95.972 |
| | 3 | 9.070 | 2.847 | 94.912 |
| | 4 | 7.077 | 3.649 | 91.231 |
| | 5 | 5.840 | 4.422 | 88.440 |
| | 6 | 4.905 | 5.265 | 87.758 |
| | 7 | 4.363 | 5.919 | 84.553 |
| | 8 | 3.842 | 6.722 | 84.029 |

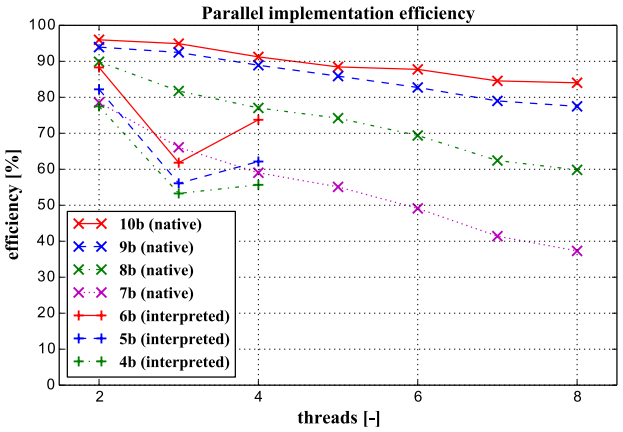



**Figure 6: Parallel implementation efficiency.**

the results. For each input data width, 100 independent runs were performed and the mean evolution time has been calculated.

The interpreted parallel implementation is limited to $\lambda$ threads since each thread treats its own individual. Besides that, the number of threads should divide $\lambda$ for the best load balancing. This is not the case of the native parallel implementation, however, sufficiently many test vectors need to be evaluated to fully utilize all eight cores of the E5-2670 processor (Figure 6).

The parallel efficiency is affected by the dynamic frequency scaling which is present in Intel's processors. In real deployment, when all processor cores are fully loaded, the sequential implementation evinces slightly worse performance due to lower frequency, hence the parallel efficiency is better. Especially on multi-socket systems, one must mind the affinity of the threads, preferably by binding individual threads to specific processor cores, to reduce cache misses and keep the memory access time as low as possible.

## 5.3 Test problems

Three different configurations of the evolutionary algorithm have been examined – standard single-population CGP optionally parallelized, multi-population CGP with few isolated islands and massively parallel CGP exploiting tens of islands. The performance of these approaches has been evaluated in the task of a combinational adder and multiplier design, in the literature generally considered as very difficult tasks [23, 18]. The evolutionary design was stopped after finding a fully functional solution.

**Table 4: Combinational adder design performance.**

| width | nodes $n_r \times n_c$ | hosts/ threads | time [s] mean | median | std |
|---|---|---|---|---|---|
| $6 \times 6$ | $1 \times 100$ | 1/6 | 39.4644 | 31.713 | 27.87117 |
| | | 6/1 | 12.3851 | 10.620 | 7.966 |
| | | 60/1 | 3.259 | 2.871 | 1.608 |
| $7 \times 7$ | $1 \times 150$ | 1/6 | 142.419 | 103.722 | 139.862 |
| | | 6/1 | 53.479 | 44.286 | 37.883 |
| | | 60/1 | 17.569 | 16.410 | 7.864 |
| $8 \times 8$ | $1 \times 200$ | 1/6 | 367.915 | 307.545 | 254.455 |
| | | 6/1 | 129.546 | 106.111 | 76.887 |
| | | 60/1 | 57.961 | 48.986 | 44.527 |
| $9 \times 9$ | $1 \times 250$ | 1/6 | 3085.891 | 2802.061 | 1548.292 |
| | | 6/1 | 1607.212 | 1413.261 | 795.004 |
| | | 60/1 | 525.032 | 462.648 | 250.616 |

**Table 5: Combinational multiplier design performance.**

| width | nodes $n_r \times n_c$ | hosts/ threads | time [s] mean | median | std |
|---|---|---|---|---|---|
| $2 \times 2$ | $1 \times 50$ | 1/1 | 0.00451 | 0.00333 | 0.00385 |
| $3 \times 2$ | $1 \times 100$ | 1/1 | 0.0451 | 0.0338 | 0.0319 |
| $3 \times 3$ | $1 \times 200$ | 1/1 | 1.897 | 1.469 | 1.605 |
| | | 6/1 | 0.530 | 0.417 | 0.403 |
| $4 \times 3$ | $1 \times 400$ | 1/4 | 10.365 | 8.427 | 8.726 |
| | | 6/1 | 5.106 | 3.979 | 3.991 |
| | | 60/1 | 2.457 | 2.210 | 1.140 |
| $4 \times 4$ | $1 \times 800$ | 1/4 | 817.689 | 874.075 | 148.275 |
| | | 6/1 | 538.058 | 458.494 | 310.345 |
| | | 60/1 | 191.175 | 154.922 | 141.206 |
| $5 \times 4$ | $1 \times 1200$ | 60/1 | 761.327 | 700.151 | 303.906 |
| $5 \times 5$ | $1 \times 1600$ | 60/2 | 16452.75[3] | | |

Table 4 shows the statistics for combinational adders of data widths 6–9 bits calculated on a set of 100 independent runs for each experimental setup, namely the mean evolution time, median and standard deviation. In the case of the multi-population approaches, the migration of the best individuals occurred every 100 000 generations. It can be observed that the multi-population approach even with few isolated islands significantly reduces the time requirements on the design process compared to the single-population CGP using the same computational capacity (6 threads vs. 6 processes). By increasing the number of islands, the evolution time decreases and according to the standard deviation, the convergence becomes more stable. The stalling effect in the fitness function, commonly observed when using other approaches [18, 1], is mitigated as a consequence of a more explorative search.

The evolutionary design of combinational multipliers is an even more complex task. No satisfactory results related to techniques operating on the whole circuit without decomposition have been published so far. While paper [23] reports only a single complete run for the 4-bit multiplier, with the aid of the proposed highly optimized CGP implementation, we can routinely design 4-bit multipliers and moreover, 5-bit multipliers are feasible as well (Table 5).

# 6. CONCLUSIONS

In this paper, highly optimized CGP implementations have been presented. Starting with several sequential versions, the paper thoroughly examines miscellaneous implementation aspects and gives detailed performance comparisons of the proposed approaches. Parallelism at various levels has been applied in order to speed up the evolutionary design process. The native implementation based on compilation of the genotype into machine code exploits the instruction-level parallelism by reducing program branching and increasing the arithmetic intensity. A large amount of test vectors can be evaluated in parallel thanks to the use of AVX instructions. Besides a thread-parallel version, a process-parallel implementation based on the isolated islands model has been proposed.

The performance and scalability has been demonstrated on the task of combinational adders and multipliers design which is believed to be a very complex task. No additional knowledge has been introduced into the design process. All

[3]Due to the very high computational effort, only a single experiment has been executed for the 5-bit multiplier.

experiments started from a randomly generated initial population. In comparison with the previously published results regarding similar evolutionary design approaches, much more complex circuits are feasible to be designed with the proposed CGP implementation.

Note that the absence of a crossover operator in CGP is a potential limiting factor and by inventing a suitable one, more efficient parallel evolutionary approaches could be applied. In our future research, we will focus on investigating more sophisticated spatially structured evolutionary algorithms with the aim of designing even more complex combinational circuits on computer clusters.

## 7. ACKNOWLEDGMENTS

This work was supported by the Czech Science Foundation project 14-04197S. The access to the CERIT-SC computing and storage facilities provided under the programme Center CERIT Scientific Cloud, part of the Operational Program Research and Development for Innovations, reg. no. CZ. 1.05/3.2.00/08.0144 is highly appreciated.

## 8. REFERENCES

[1] T. Aoki, N. Homma, and T. Higuchi. Evolutionary synthesis of arithmetic circuit structures. *Artif. Intell. Rev.*, 20(3-4):199–232, Dec. 2003.

[2] E. Cantu-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[3] J. Clegg, J. A. Walker, and J. F. Miller. A new crossover technique for cartesian genetic programming. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1580–1587, London, 7-11 July 2007. ACM Press.

[4] A. Fox. Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for intel, amd and via cpus. `http://agner.org/optimize/instruction_tables.pdf`, 2013.

[5] B. W. Goldman and W. F. Punch. Reducing wasted evaluations in cartesian genetic programming. In *Proceedings of the 16th European Conference on Genetic Programming*, EuroGP'13, pages 61–72, Berlin, Heidelberg, 2013. Springer-Verlag.

[6] S. Harding and W. Banzhaf. Implementing cartesian genetic programming classifiers on graphics processing units using gpu.net. In *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Companion Material Proceedings, Dublin, Ireland, July 12-16, 2011*, pages 463–470. ACM, 2011.

[7] S. Harding, J. F. Miller, and W. Banzhaf. Self modifying cartesian genetic programming: Parity. In *2009 IEEE Congress on Evolutionary Computation*, pages 285–292. IEEE Press, 2009.

[8] J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach*. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science, 2011.

[9] R. Hrbacek and M. Sikulova. Coevolutionary cartesian genetic programming in fpga. In *Advances in Artificial Life, ECAL 2013, Proceedings of the Twelfth European Conference on the Synthesis and Simulation of Living Systems*, pages 431–438. MIT Press, 2013.

[10] J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, and G. Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, 2003.

[11] J. Miller and S. Smith. Redundancy and computational efficiency in cartesian genetic programming. *Evolutionary Computation, IEEE Transactions on*, 10(2):167–174, 2006.

[12] J. Miller and P. Thomson. Cartesian genetic programming. In *Genetic Programming*, volume 1802 of *Lecture Notes in Computer Science*, pages 121–132. Springer Berlin Heidelberg, 2000.

[13] J. F. Miller, editor. *Cartesian Genetic Programming*. Natural Computing Series. Springer Verlag, 2011.

[14] R. Poli and J. Page. Solving high-order boolean parity problems with smooth uniform crossover, sub-machine code gp and demes. *Genetic Programming and Evolvable Machines*, 1(1-2):37–56, Apr. 2000.

[15] R. Seyfarth. *Introduction to 64 Bit Intel Assembly Language Programming for Linux*. CreateSpace Independent Publishing Platform, 2012.

[16] A. P. Shanthi and R. Parthasarathi. Practical and scalable evolution of digital circuits. *Appl. Soft Comput.*, 9(2):618–624, Mar. 2009.

[17] D. Sherry, K. Veeramachaneni, J. McDermott, and U.-M. O'Reilly. Flex-gp: Genetic programming on the cloud. In *Applications of Evolutionary Computation - EvoApplications 2012*, volume 7248 of *LNCS*, pages 477–486. Springer, 2012.

[18] E. Stomeo, T. Kalganova, and C. Lambert. Generalized disjunction decomposition for evolvable hardware. *Trans. Sys. Man Cyber. Part B*, 36(5):1024–1043, Oct. 2006.

[19] M. Tomassini. *Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time (Natural Computing Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[20] Z. Vasicek and L. Sekanina. Hardware accelerator of cartesian genetic programming with multiple fitness units. *Computing and Informatics*, 29(6):1359–1371, 2010.

[21] Z. Vasicek and L. Sekanina. Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. *Genetic Programming and Evolvable Machines*, 12(3):305–327, 2011.

[22] Z. Vasicek and K. Slany. Efficient phenotype evaluation in cartesian genetic programming. In *Proc. of the 15th European Conference on Genetic Programming*, LNCS 7244, pages 266–278. Springer Verlag, 2012.

[23] V. K. Vassilev, D. Job, and J. F. Miller. Towards the automatic design of more efficient digital circuits. *Evolvable Hardware, NASA/DoD Conference on*, 0:151, 2000.

[24] T. Yu and J. Miller. Finding needles in haystacks is not hard with neutrality. In *Genetic Programming*, volume 2278 of *Lecture Notes in Computer Science*, pages 13–25. Springer Berlin Heidelberg, 2002.

## A.2 Bent Function Synthesis by Means of Cartesian Genetic Programming

Radek Hrbacek and Vaclav Dvorak

In *Parallel Problem Solving from Nature - PPSN XIII*.
PPSN 2014, Heidelberg, DE, Springer Verlag, 2014.
ISBN: 978-3-319-10761-5, pp. 414-423.

# Bent Function Synthesis by Means of Cartesian Genetic Programming

Radek Hrbacek and Vaclav Dvorak

Brno University of Technology,
Faculty of Information Technology
Bozetechova 2, 61266 Brno, Czech Republic
ihrbacek@fit.vutbr.cz, dvorak@fit.vutbr.cz
http://www.fit.vutbr.cz/~ihrbacek/

**Abstract.** In this paper, a new approach to synthesize bent Boolean functions by means of Cartesian Genetic Programming (CGP) is proposed. Bent functions have important applications in cryptography due to their high nonlinearity. However, they are very rare and their discovery using conventional brute force methods is not efficient enough. We show that by using CGP we can routinely design bent functions of up to 16 variables. The evolutionary approach exploits parallelism in both the fitness calculation and the search algorithm.

## 1 Introduction

Evolutionary Algorithms (EAs) have been recently used in many engineering areas as design and optimization methods. Thanks to the innovation introduced into the design process, they are able to outperform conventional approaches in particular problems. Several types of EAs have been successfully employed in the task of evolutionary circuit design. Besides Genetic Programming (GP) heavily used by John Koza [1] to automatically design analog circuits, regulators, optical systems or antennas, excellent results have been achieved with the use of Cartesian Genetic Programming (CGP) [2]. The applications of CGP include combinational circuits design [3] and optimization [4], digital image filter design [5, 6], artificial neural networks design [7] and many others.

The evolutionary design is often very computationally demanding approach. In order to reduce the design time, various application specific accelerators as well as evolutionary algorithm modifications have been proposed. While the former case typically involves parallel fitness function implementation based on FPGA accelerators [6, 8] or running on multicore CPUs, GPUs [9] or even computer clusters [3] and exploiting parallelism at various levels (instruction, data, thread or process), the latter one includes genotype representation or search algorithm modifications. In the past, spatially structured evolutionary algorithms have been intensively studied and variety of approaches differing in the used evolutionary algorithm or communication topology has emerged [10–12].

While the use of computers and communication networks is becoming more and more popular, one has to seriously deal with the security of the data being

stored or transferred. In cryptography, the two most fundamental techniques to achieve security in systems are *confusion* and *diffusion* [13]. Confusion refers to making a complex relationship between the ciphertext and the key. Thanks to diffusion, the statistical structure of the plain text is dissipated over significant part of the ciphertext, which prevents from reconstructing the original statistical information. In real cryptographic systems, the cipher key is much shorter than the message being encrypted and thus the key has to be reused in some way, often by applying a Boolean function to the key all over again. To avoid decryption by an attacker, the key sequence has to be random. If the Boolean function used to generate the key stream is close to linear, the message can be possibly deciphered. By using functions that are as far from linear as possible, one can build more secure cryptographic systems [14]. These functions, called *bent* functions, are very rare.

The state of the art methods for finding them operate usually on the brute force principle although exploiting some properties of the functions in order to reduce the size of the search space [15]. The number of Boolean functions grows exponentially with the number of variables, while the relative frequency of bent functions decreases. Therefore, for higher number of variables (the literature reports only functions of no more than 8 variables), these methods are not efficient enough. Another approach based on genetic algorithm (GA) is very limited as well. Even though the GA seems to be suitable for this purpose, the proposed approach is not scalable enough [16].

Inspired by the evolutionary design of combinational circuits by means of CGP, we propose a CGP based synthesis of bent Boolean functions. The parallelism at the data, thread and process level has been applied in order to take advantage of modern processor architectures and computer clusters. The scalability of this approach has been earlier verified in the task of evolutionary design of combinational adders and multipliers [3].

The paper is organized as follows. Section 2 introduces bent Boolean functions from the mathematical perspective. CGP is discussed in Section 3 and the proposed evolutionary approach to synthesize bent functions is described in Section 4. Section 5 is dedicated to experiments and the achieved results, final conclusions can be found in Section 6.

## 2 Bent Boolean functions

Boolean functions are of great importance for various cryptographic algorithms. Special attention is paid to the design of nonlinear Boolean functions due to their resistance to linear cryptanalysis [17]. This section presents necessary mathematical definitions for the purpose of introduction of bent functions [14, 15].

**Definition 1.** *A* ***Boolean function*** *is a function of the form* $f : \mathrm{D}^n \to \mathrm{D}$*, where* $\mathrm{D} = \{0, 1\}$ *is a Boolean domain and* $n \geq 0$ *is the arity of the function. For a function* $f$*, let* $f_0 = f(0, 0, \ldots, 0)$*,* $f_1 = f(0, 0, \ldots, 1)$*, ...,* $f_{2^n-1} = f(1, 1, \ldots, 1)$*.* $\mathrm{TT}_f = (f_{2^n-1} \cdots f_1 f_0)$ *is the* ***truth table representation*** *of the function* $f$*.*

**Definition 2.** *A **linear** (Boolean) function is either the constant 0 function or the exclusive OR (XOR) of one or more variables. An **affine** (Boolean) function is a linear function or the complement of a linear function.*

**Definition 3.** *The **Hamming distance** $d(f, g)$ between two functions $f$ and $g$ is the number of truth table entries with different values.*

**Definition 4.** *The **nonlinearity** $\mathrm{NL}_f$ of a function $f$ is the minimum Hamming distance between the function $f$ and an affine function.*

**Definition 5.** *Let $f$ be a Boolean function of even arity $n$, $f$ is a **bent function** iff its nonlinearity $\mathrm{NL}_f$ is maximum among $n$-variable functions.*

Affine functions are not suitable for the use in cryptography, since they are susceptible to a linear attack. Therefore, one seeks functions that are as far away (in the Hamming distance) as possible from all the affine functions – these are the bent functions. The nonlinearity of a bent function $f$ of $n$ variables is $\mathrm{NL}_f = 2^{n-1} - 2^{\frac{n}{2}-1}$ [18]. This constraint is not applicable for functions of odd

**Table 1.** Examples of 4-variable Boolean functions and their nonlinearities.

| | function $f$ | truth table $\mathrm{TT}_f$ | nonlinearity $\mathrm{NL}_f$ |
|---|---|---|---|
| linear | 0 | 0000000000000000 | 0 |
| | $x_0$ | 1010101010101010 | 0 |
| | $x_1$ | 1100110011001100 | 0 |
| | $x_1 \oplus x_0$ | 0110011001100110 | 0 |
| | $x_2$ | 1111000011110000 | 0 |
| | $x_2 \oplus x_0$ | 0101101001011010 | 0 |
| | $x_2 \oplus x_1$ | 0011110000111100 | 0 |
| | $x_2 \oplus x_1 \oplus x_0$ | 1001011010010110 | 0 |
| | $x_3$ | 1111111100000000 | 0 |
| | $x_3 \oplus x_0$ | 0101010110101010 | 0 |
| | $x_3 \oplus x_1$ | 0011001111001100 | 0 |
| | $x_3 \oplus x_1 \oplus x_0$ | 1001100101100110 | 0 |
| | $x_3 \oplus x_2$ | 0000111111110000 | 0 |
| | $x_3 \oplus x_2 \oplus x_0$ | 1010010101011010 | 0 |
| | $x_3 \oplus x_2 \oplus x_1$ | 1100001100111100 | 0 |
| | $x_3 \oplus x_2 \oplus x_1 \oplus x_0$ | 0110100110010110 | 0 |
| nonlinear | $x_3x_0$ | 1010101000000000 | 4 |
| | $x_2x_1x_1 \oplus x_3 \oplus x_0$ | 1101010100101010 | 2 |
| | $x_3x_0 \oplus x_1$ | 0110011011001100 | 4 |
| | $x_3x_2 \oplus x_1 \oplus x_0$ | 0110011011001100 | 4 |
| bent | $x_3x_2 \oplus x_1x_0$ | 0001000100011110 | 6 |
| | $x_3x_0 \oplus (x_2 \oplus x_0)x_1 \oplus x_2 \oplus x_0$ | 1011100000010010 | 6 |

arity that can, in general, have greater nonlinearity. This paper deals only with functions of even number of variables.

Examples of Boolean functions of 4 variables can be seen in Table 1. In the first 16 rows, all linear functions are listed, followed by several nonlinear and bent functions, the maximum nonlinearity of 4-variable functions is $\mathrm{NL}_f = 2^{4-1} - 2^{\frac{4}{2}-1} = 6$.

The number of different Boolean functions grows exponentially with the number of variables: $N_f(n) = 2^{2^n}$. However, the relative frequency of bent functions decreases very fast (see Table 2) and thus, for $n \geq 6$, identifying them is like looking for a needle in a haystack.

**Table 2.** Relative frequency of $n$-variable bent functions [14].

| variables $n$ | 2 | 4 | 6 | 8 |
|---|---|---|---|---|
| Boolean functions | $2^4$ | $2^{16}$ | $2^{64}$ | $2^{256}$ |
| bent functions | $2^3$ | $\approx 2^{9.8}$ | $\approx 2^{32.3}$ | $\approx 2^{106.3}$ |
| relative frequency | $2^{-1}$ | $\approx 2^{-6.2}$ | $\approx 2^{-31.7}$ | $\approx 2^{-149.7}$ |

Recently, various approaches based on the properties of bent functions have been proposed, effectively reducing the number of the Boolean functions needed to be verified in order to identify bent functions by means of a brute force search [16, 15]. In some special cases, bent functions can be constructed directly [17].

## 3 Cartesian Genetic Programming

Cartesian genetic programming - a branch of genetic programming - has been introduced by Miller [2] and since then it has been successfully applied to a number of challenging real-world problems [19]. In contrast with GP which uses tree representation, an individual in CGP is represented by a directed acyclic graph. This dissimilarity enables the candidate solution to automatically reuse intermediate results and have multiple outputs, which makes CGP very suitable for design of various kinds of digital circuits, digital filters, etc.

A candidate program in CGP consist of the cartesian grid of $n_\mathrm{r} \times n_\mathrm{c}$ programmable nodes interconnected by a feed-forward network, as it can be seen in Figure 1. Node inputs can be connected either to one of $n_\mathrm{i}$ primary inputs or to a node in preceding $l$ columns, each node has usually a fixed number of inputs $n_\mathrm{ni} = 2$. Each node can perform one of $n_\mathrm{ni}$-input functions from the set $\Gamma$. Each of $n_\mathrm{o}$ primary circuit outputs is connected either to a primary input or a node output, the output connectivity can be additionally restricted by the $o$-back parameter. By changing the grid size and the $l$-back parameter, one can control the area and delay of the circuit.

Thanks to the fixed topology of CGP programs, each chromosome can be encoded using an fixed-sized array of $n_\mathrm{r} \cdot n_\mathrm{c} \cdot (n_\mathrm{ni} + 1) + n_\mathrm{o}$ integers ($n_\mathrm{ni}$ inputs

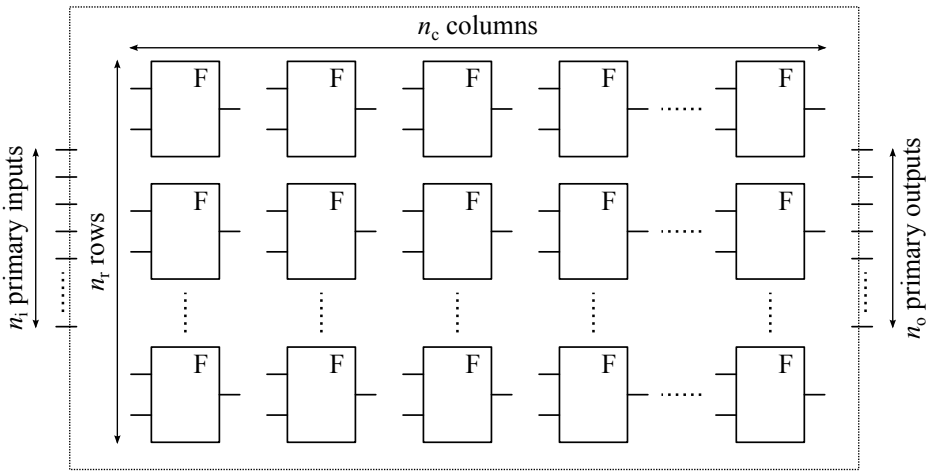



**Fig. 1.** Cartesian genetic programming scheme.

and one function per each node). Each primary input is assigned a number from $\{0, ..., n_i - 1\}$ and the nodes are assigned numbers from $\{n_i, ..., n_i + n_r \cdot n_c - 1\}$. Unlike the genotype, the phenotype is of variable length depending on the number of inactive nodes (i.e. nodes whose output is not used by any other node or primary output), which implies the existence of individuals with different genotypes but the same phenotypes. The existence of individuals with different genotypes but with the same fitness value is usually referred to as neutrality. For certain problems, the neutrality significantly reduces the computational effort and helps to find more innovative solutions [20].

CGP uses a simple mutation based $(1 + \lambda)$ evolutionary strategy as a search mechanism, the population size $1+\lambda$ is mostly very small, typically, $\lambda$ is between 1 and 15. The initial population is constructed randomly in most cases, however, it can be seeded with a known solution as well (evolutionary optimization) [4]. In each generation, the best individual or a sibling with the same fitness value is passed to the next generation unmodified along with its $\lambda$ offspring individuals created by means of point mutation operator. The mutation rate $m$ is usually set to modify up to 5 % randomly selected genes. Usually, no crossover operator is used in CGP, however, for particular problems (e.g. symbolic regression), special crossover operators have been investigated [21]. None of them has been confirmed as useful for other problem classes so far.

In the case of combinational circuit design, the fitness function is given by the number of correct output bits compared to a specified truth table. All combinations of input values ($2^{n_i}$ test vectors for a circuit with $n_i$ inputs and $n_o$ outputs) have to be fetched to the primary inputs in order to obtain a fully working circuit. $n_o \cdot 2^{n_i}$ output bits have to be verified so as to compute the fitness value.

## 4 Bent function synthesis by means of CGP

The principle of bent function synthesis by means of CGP is very similar to the case of combinational circuit design, since every Boolean function can be implemented by a combinational circuit. The difference lies in the fitness function. Unlike combinational circuits having fitness value equal to the total number of wrong output bits, the fitness value of a bent function candidate is its nonlinearity, i.e. the lowest Hamming distance from a linear function. Despite the fact, that bent Boolean functions have single output comparing to combinational circuits having arbitrary many outputs, the fitness calculation is computationally more intensive, since the number of linear functions being compared with the candidate individual grows exponentially with the number of variables.

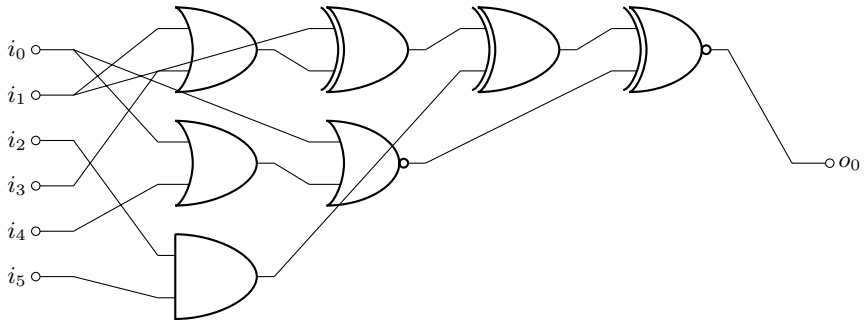



**Fig. 2.** Example of an CGP individual representing the Boolean function $f(i_5, \ldots, i_0) = o_0 = \overline{((i_1 \oplus (i_1 + i_3)) \oplus i_2 i_5) \oplus \overline{i_0 + (i_0 + i_4)}}$ with the truth table $\mathrm{TT}_f = 00111100011010010011001101100110111100001010010111111111110101010$. This function has nonlinearity $\mathrm{NL}_f = 28$ and thus it is bent.

Figure 2 depicts an example of an CGP individual representing a Boolean function. Note that the representation is not optimal in terms of area or delay, since the only significant property is the truth table.

While evaluating an individual's fitness value, all active genes of the chromosome need to be traversed and their output values need to be calculated. The single output is then compared against all linear functions simply by `XOR`ing the values and counting the number of ones. There is no need to compare the values to the remaining affine functions (the complements of linear functions), since the following always holds true:

$$d(f, g) + d(f, g_c) = 2^n, \tag{1}$$

where $f, g$ are arbitrary $n$-variable Boolean functions and $g_c$ is complementary to $g$.

The entire evolutionary design process can be accelerated in the same way as it has been done in the case of combinational circuits [3]. The test vectors

can be fed to the CGP individual in parallel, from 64 test vectors within a standard x86-64 register up to 256 test vectors using AVX extension. Moreover, the population can be split over a number of threads, each thread handling a portion of the population. Nevertheless, the number of threads is substantially limited by the population size, which is usually very small in CGP. In order to take advantage of multicore processors or even computer clusters, additional level of parallelism has to be exploited. By introducing spatially structured EA principle, one can scale the evolutionary process onto arbitrary sized computer cluster. Unfortunately, the absence of crossover operator in CGP is a very limiting factor, since most parallel algorithms are based on combining genotypes from different spatially isolated populations. Thus, simple isolated islands model with periodical exchange of the best individual is used [3].

## 5 Experimental results

In this section, experiments regarding the ability of the proposed approach to synthesize bent functions are presented. All the experiments were performed on a computer cluster of 112 nodes with the following hardware configuration: 2× 8-core Intel E5-2670, 128 GB RAM, 2× 600 GB 15 k scratch hard disks, connected by gigabit Ethernet and Infiniband links.

The performance of the CGP based approach has been examined in terms of the evolution time. The CGP parameters were set as follows on the basis of previous experiments with combinational circuits [3]: the functions set $\Gamma = \{\texttt{BUF}, \texttt{NOT}, \texttt{AND}, \texttt{OR}, \texttt{XOR}, \texttt{NAND}, \texttt{NOR}, \texttt{XNOR}\}$, population of 5 individuals, mutation

**Table 3.** Bent Boolean functions CGP based synthesis times.

| $n$ | nodes $n_r \times n_c$ | hosts/ threads | time [s] | | |
|---|---|---|---|---|---|
| | | | mean | median | std |
| 6 | $1 \times 50$ | 1/1 | 0.000819 | 0.000685 | 0.000668 |
| 8 | $1 \times 100$ | 1/1 | 0.00470 | 0.00343 | 0.00410 |
| 10 | $1 \times 150$ | 1/1 | 0.0602 | 0.0442 | 0.0483 |
| 12 | $1 \times 200$ | 1/1 | 2.0443 | 1.4057 | 1.9579 |
| | | 1/4 | 1.1291 | 0.8392 | 1.0610 |
| | | 4/1 | 0.8240 | 0.6267 | 0.5405 |
| | | 40/1 | 0.3859 | 0.3618 | 0.1080 |
| 14 | $1 \times 250$ | 1/1 | 133.202 | 91.765 | 146.839 |
| | | 1/4 | 76.040 | 54.954 | 72.808 |
| | | 4/1 | 44.680 | 35.700 | 34.165 |
| | | 40/1 | 15.806 | 15.255 | 4.853 |
| 16 | $1 \times 300$ | 1/1 | 6223.66 | 4666.82 | 4734.02 |
| | | 1/4 | 3880.06 | 3744.23 | 2571.49 |
| | | 4/1 | 1855.79 | 1543.12 | 1329.10 |
| | | 40/1 | 636.13 | 565.68 | 229.06 |

rate 5 %. The number of rows was set to $n_r = 1$ and the $l$-back parameter was maximal, enabling the greatest connectibility (there were no requirements on the propagation delay). The size of the grid was empirically chosen for each variable count $n$ as a optimal choice with respect to the evolution time (about 10× larger than the average individual found). No limitations on the number of generations were imposed, each run was successful. The spatially structured implementations exchanged the best individual over all populations every 100 generations.

The achieved results can be seen in Table 3, four different configurations of the algorithm were compared – basic single threaded variant, accelerated 4-thread parallel version, 4-island and 40-island spatially structured variants. For each configuration, 100 independent runs were performed and common statistical metrics were calculated – the mean time, the median value and the standard deviation. The evolution times for functions of less than 12 variables are negligible and cannot be improved by means of thread or process level parallelism, because there is not enough work to distribute. For higher numbers of variables, the computational effort grows rapidly and the parallel implementations help significantly to reduce the evolution time. For example, the design of 16-variable bent functions can be sped up 10× on the computer cluster in comparison with the basic single threaded implementation. It shows that even a small number of isolated populations can more efficiently utilize the power of a multicore processor than the multithreaded single population approach. Not only the mean and median times, but also the standard deviations of the evolution times are lower, increasing the probability of finding a bent function in a limited time.

An example of a bent Boolean function of 16 variables synthesized by means of CGP can be seen in Figure 3. Its nonlinearity is 32,640 and the CGP representation has 24 active nodes with the maximum delay of 7.

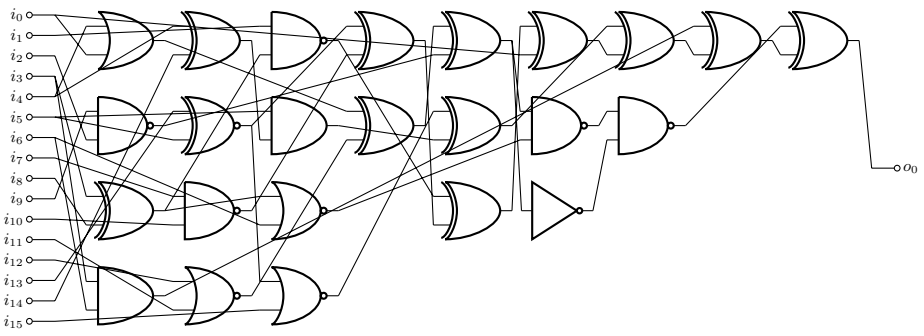



**Fig. 3.** CGP representation of a 16-variable bent Boolean function.

## 6 Conclusions

In this paper, a new approach to synthesize bent Boolean functions based on CGP has been proposed. Bent functions have applications in cryptography due to their significant properties – when used in a cipher, their nonlinearity makes cryptanalysis harder. The relative frequency of bent functions among all Boolean functions of the same arity is rapidly decreasing with the number of variables and designing such functions is harder and harder.

It was shown, that by using CGP, we are able to routinely design bent Boolean functions of up to 16 variables. The evolutionary process was sped up by employing various levels of parallelism in both fitness calculation and the search algorithm, which gives a great scalability to the proposed approach. Several algorithm configurations were experimentally compared and it was shown, that by using a simple isolated island model, one can significantly reduce the evolution time. Additional effort has to be made in order to investigate potential common features shared by bent functions found using independent CGP runs.

Even though bent Boolean functions themselves have great properties, in order to achieve maximum confusion in real cryptographic systems, there should be a balance between bits that are changed and that are not. This can be achieved by using *balanced* functions; however, no bent function is balanced and thus a trade-off between nonlinearity and balance has to be sought [17, 14]. In our future research, we want to focus on designing such functions by means of evolutionary algorithms. Further work will be also devoted to the optimization of the synthesized functions in terms of area and delay inspired by fast SAT-based optimization methods for complex combinational circuits [4].

## 7 Acknowledgements

This work was supported by the Czech Science Foundation project 14-04197S. The access to the CERIT-SC computing and storage facilities provided under the programme Center CERIT Scientific Cloud, part of the Operational Program Research and Development for Innovations, reg. no. CZ. 1.05/3.2.00/08.0144 is appreciated.

## References

1. Koza, J.R.: Genetic Programming IV: Routine Human-Competitive Machine Intelligence. Kluwer Academic Publishers, Norwell, MA, USA (2003)
2. Miller, J., Thomson, P.: Cartesian genetic programming. In: Genetic Programming. Volume 1802 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2000) 121–132
3. Hrbacek, R., Sekanina, L.: Towards highly optimized cartesian genetic programming: From sequential via simd and thread to massive parallel implementation. In: Proceeding of Genetic and Evolutionary Computation Conference, GECCO 2014, Association for Computing Machinery (2014) (to appear).

4. Vasicek, Z., Sekanina, L.: On area minimization of complex combinational circuits using cartesian genetic programming. In: 2012 IEEE World Congress on Computational Intelligence, Institute of Electrical and Electronics Engineers (2012) 2379–2386
5. Vasicek, Z., Bidlo, M.: Evolutionary design of robust noise-specific image filters. In: 2011 IEEE Congress on Evolutionary Computation, IEEE Computer Society (2011) 269–276
6. Hrbacek, R., Sikulova, M.: Coevolutionary cartesian genetic programming in fpga. In: Advances in Artificial Life, ECAL 2013, Proceedings of the Twelfth European Conference on the Synthesis and Simulation of Living Systems, MIT Press (2013) 431–438
7. Khan, G., Miller, J.: The cgp developmental network. In Miller, J.F., ed.: Cartesian Genetic Programming. Natural Computing Series. Springer Berlin Heidelberg (2011) 255–291
8. Vasicek, Z., Sekanina, L.: Hardware accelerator of cartesian genetic programming with multiple fitness units. Computing and Informatics **29**(6) (2010) 1359–1371
9. Harding, S., Banzhaf, W.: Hardware acceleration for cgp: Graphics processing units. In Miller, J.F., ed.: Cartesian Genetic Programming. Natural Computing Series. Springer Berlin Heidelberg (2011) 231–253
10. Cantu-Paz, E.: Efficient and Accurate Parallel Genetic Algorithms. Kluwer Academic Publishers, Norwell, MA, USA (2000)
11. Tomassini, M.: Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time (Natural Computing Series). Springer-Verlag New York, Inc., Secaucus, NJ, USA (2005)
12. Jaros, J.: Multi-gpu island-based genetic algorithm solving the knapsack problem. In: 2012 IEEE World Congress on Computational Intelligence, Institute of Electrical and Electronics Engineers (2012) 217–224
13. Shannon, C.: Communication theory of secrecy systems. Bell System Technical Journal, Vol 28, pp. 656715 (Oktober 1949)
14. Butler, J.T., Sasao, T.: Logic functions for cryptography - a tutorial. In: Proceedings of the Reed-Muller Workshop. (2009)
15. Shafer, J.L., Schneider, S.W., Butler, J.T., Stanica, P.: Enumeration of bent boolean functions by reconfigurable computer. In Sass, R., Tessier, R., eds.: FCCM, IEEE Computer Society (2010) 265–272
16. Schneider, S.W.: Finding bent functions using genetic algorithms. Master's thesis, Naval Postgraduate School, Monterey (2009)
17. Dobbertin, H.: Construction of bent functions and balanced boolean functions with high nonlinearity. In Preneel, B., ed.: Fast Software Encryption. Volume 1008 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1995) 61–74
18. Rothaus, O.: On "bent" functions. Journal of Combinatorial Theory, Series A **20**(3) (1976) 300 – 305
19. Miller, J.F., ed.: Cartesian Genetic Programming. Natural Computing Series. Springer Verlag (2011)
20. Miller, J., Smith, S.: Redundancy and computational efficiency in cartesian genetic programming. Evolutionary Computation, IEEE Transactions on **10**(2) (2006) 167–174
21. Clegg, J., Walker, J.A., Miller, J.F.: A new crossover technique for cartesian genetic programming. In: GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation. Volume 2., London, ACM Press (7-11 July 2007) 1580–1587

## A.3 Bent Functions Synthesis on Intel Xeon Phi Coprocessor

Radek Hrbacek

In *Mathematical and Engineering Methods in Computer Science*.
MEMICS 2014, Heidelberg, DE, Springer Verlag, 2011.
ISBN: 978-3-319-14895-3, pp. 88-99.

# Bent Functions Synthesis on Intel Xeon Phi Coprocessor

Radek Hrbacek

Brno University of Technology,
Faculty of Information Technology
Bozetechova 2, 61266 Brno, Czech republic
ihrbacek@fit.vutbr.cz
http://www.fit.vutbr.cz/ ihrbacek/

**Abstract.** A new approach to synthesize bent Boolean functions by means of Cartesian Genetic Programming (CGP) has been proposed recently. Bent functions have important applications in cryptography due to their high nonlinearity. However, they are very rare and their discovery using conventional brute force methods is not efficient enough. In this paper, a new parallel implementation is proposed and the performance is evaluated on the Intel Xeon Phi Coprocessor.

## 1 Introduction

Evolutionary algorithms (EA) have been recently successfully applied to a number of challenging real-world problems as design and optimization methods. Several types of EAs have been successfully employed in the task of evolutionary circuit design, excellent results have been achieved with the use of Cartesian Genetic Programming (CGP) [1]. CGP has been used to design and optimize combinational circuits [2, 3], digital image filters [4, 5], artificial neural networks [6] and many others.

Recently, CGP has been applied to synthesize bent Boolean functions [7]. It was shown, that by using CGP, it is possible to routinely design bent Boolean functions of up to 16 variables. The evolutionary process was sped up by employing various levels of parallelism in both fitness calculation and the search algorithm, however, the computational demands were still high.

A continued rise in transistor density allowed the processor manufacturers to integrate more processor cores; recently, a new *many-core* architectural concept emerged. An example of this approach, the Intel Xeon Phi coprocessor, promises to reach high performance without the need to change the programming model, at least for well parallelizable problems. We propose a new approach to speed up the fitness computation by exploiting the massive parallelism of the Xeon Phi coprocessor.

The paper is organized as follows. Section 2 introduces bent Boolean functions from the mathematical perspective. The principles of evolutionary design of bent functions and the new approach are discussed in Section 3. Section 4 deals with the Intel Xeon Phi coprocessor. Section 5 is dedicated to experiments and the achieved results, final conclusions can be found in Section 6.

## 2 Bent Boolean functions

Boolean functions are of great importance for various cryptographic algorithms. Special attention is paid to nonlinear Boolean functions, because their use can increase the resistance to linear cryptanalysis [8]. This section presents necessary mathematical definitions to introduce bent Boolean functions [9, 10].

**Definition 1** *A **Boolean function** is a function of the form* $f : \mathrm{D}^n \to \mathrm{D}$, *where* $\mathrm{D} = \{0, 1\}$ *is a Boolean domain and* $n \geq 0$ *is the arity of the function. For a function* $f$, *let* $f_0 = f(0, 0, \ldots, 0)$, $f_1 = f(0, 0, \ldots, 1)$, ..., $f_{2^n-1} = f(1, 1, \ldots, 1)$. $\mathrm{TT}_f = (f_{2^n-1} \cdots f_1 f_0)$ *is the **truth table representation** of the function* $f$.

**Definition 2** *A **linear** (Boolean) function is either the constant 0 function or the exclusive OR (XOR) of one or more variables. An **affine** (Boolean) function is a linear function or the complement of a linear function.*

**Definition 3** *The **Hamming distance*** $d(f, g)$ *between two functions* $f$ *and* $g$ *is the number of truth table entries with different values.*

**Table 1.** Examples of 4-variable Boolean functions and their nonlinearities.

| | function $f$ | truth table $\mathrm{TT}_f$ | nonlinearity $\mathrm{NL}_f$ |
|---|---|---|---|
| linear | $0$ | 0000000000000000 | 0 |
| | $x_0$ | 1010101010101010 | 0 |
| | $x_1$ | 1100110011001100 | 0 |
| | $x_1 \oplus x_0$ | 0110011001100110 | 0 |
| | $x_2$ | 1111000011110000 | 0 |
| | $x_2 \oplus x_0$ | 0101101001011010 | 0 |
| | $x_2 \oplus x_1$ | 0011110000111100 | 0 |
| | $x_2 \oplus x_1 \oplus x_0$ | 1001011010010110 | 0 |
| | $x_3$ | 1111111100000000 | 0 |
| | $x_3 \oplus x_0$ | 0101010110101010 | 0 |
| | $x_3 \oplus x_1$ | 0011001111001100 | 0 |
| | $x_3 \oplus x_1 \oplus x_0$ | 1001100101100110 | 0 |
| | $x_3 \oplus x_2$ | 0000111111110000 | 0 |
| | $x_3 \oplus x_2 \oplus x_0$ | 1010010101011010 | 0 |
| | $x_3 \oplus x_2 \oplus x_1$ | 1100001100111100 | 0 |
| | $x_3 \oplus x_2 \oplus x_1 \oplus x_0$ | 0110100110010110 | 0 |
| nonlinear | $x_3x_0$ | 1010101000000000 | 4 |
| | $x_2x_1x_1 \oplus x_3 \oplus x_0$ | 1101010100101010 | 2 |
| | $x_3x_0 \oplus x_1$ | 0110011011001100 | 4 |
| | $x_3x_2 \oplus x_1 \oplus x_0$ | 0110011011001100 | 4 |
| bent | $x_3x_2 \oplus x_1x_0$ | 0001000100011110 | 6 |
| | $x_3x_0 \oplus (x_2 \oplus x_0)x_1 \oplus x_2 \oplus x_0$ | 1011100000010010 | 6 |

**Definition 4** *The* ***nonlinearity*** $\mathrm{NL}_f$ *of a function* $f$ *is the minimum Hamming distance between the function* $f$ *and all affine functions.*

**Definition 5** *Let* $f$ *be a Boolean function of even arity* $n$*,* $f$ *is a* ***bent function*** *iff its nonlinearity* $\mathrm{NL}_f$ *is maximum among* $n$*-variable functions.*

Affine functions are not suitable for the use in cryptography, since their usage leads to a linear attack vulnerability [8]. Therefore, we seek functions that are as far away (in the Hamming distance) as possible from all the affine functions. These are the bent functions, their nonlinearity is $\mathrm{NL}_f = 2^{n-1} - 2^{\frac{n}{2}-1}$, where $n$ variables [11].

Examples of Boolean functions of 4 variables can be seen in Table 1, the linear functions are listed in the first 16 rows, followed by several nonlinear and bent functions. The maximum nonlinearity of 4-variable functions is $\mathrm{NL}_f = 2^{4-1} - 2^{\frac{4}{2}-1} = 6$.

The number of different Boolean functions grows exponentially with the number of variables: $N_f(n) = 2^{2^n}$. However, the relative frequency of bent functions decreases very fast (see Table 2).

**Table 2.** Relative frequency of $n$-variable bent functions [9].

| variables $n$ | 2 | 4 | 6 | 8 |
|---|---|---|---|---|
| Boolean functions | $2^4$ | $2^{16}$ | $2^{64}$ | $2^{256}$ |
| bent functions | $2^3$ | $\approx 2^{9.8}$ | $\approx 2^{32.3}$ | $\approx 2^{106.3}$ |
| relative frequency | $2^{-1}$ | $\approx 2^{-6.2}$ | $\approx 2^{-31.7}$ | $\approx 2^{-149.7}$ |

Recently, various approaches to find bent functions based on the brute force search method have been proposed [12, 10]. In some special cases, bent functions can be constructed directly [8]. The evolutionary design has been shown for bent functions with up to 16 variables [7].

## 3 Evolutionary design of bent functions

The evolutionary design of bent functions is based on Cartesian genetic programming, it is very similar to the combinational circuits design [2, 7]. This section deals with the main principles of bent functions synthesis by means of CGP and introduces a new approach to efficient fitness computation.

### 3.1 Cartesian Genetic Programming

Cartesian genetic programming has been introduced by Miller [1], since then a lot of challenging problems have been solved by means of CGP [13]. Unlike GP which uses tree representation, an individual in CGP is represented by a

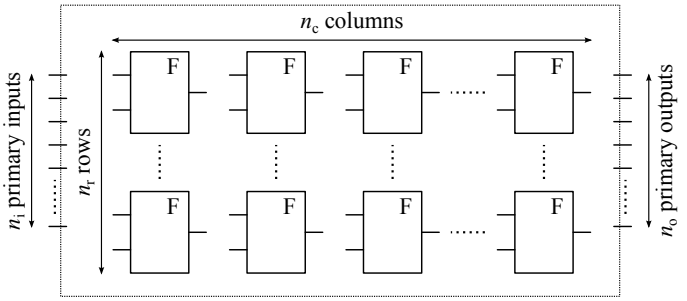



**Fig. 1.** Cartesian genetic programming scheme.

directed acyclic graph, which enables the candidate solution to automatically reuse intermediate results and have multiple outputs. Thanks to that, CGP is very suitable for the design of digital circuits.

A candidate program in CGP is composed of the cartesian grid of $n_r \times n_c$ programmable nodes interconnected by a feed-forward network (see Figure 1). Node inputs can be connected either to one of $n_i$ primary inputs or to an output of a node in preceding $l$ columns. Each node has usually a fixed number of inputs $n_{ni} = 2$ and can perform one of $n_{ni}$-input functions from the set $\Gamma$. Each of $n_o$ primary circuit outputs can be connected either to a primary input or to a node output. By changing the grid size and the $l$-back parameter, one can constrain the area and delay of the circuit.

The fixed topology of CGP programs allows to use a fixed-sized array of $n_r \cdot n_c \cdot (n_{ni} + 1) + n_o$ integers to encode the chromosome. Each primary input is assigned a number from $\{0, ..., n_i - 1\}$ and the nodes are assigned numbers from $\{n_i, ..., n_i + n_r \cdot n_c - 1\}$. The phenotype is of variable size depending on the number of active nodes (i.e. nodes which are necessary to compute the primary outputs), which implies the existence of individuals with different genotypes but the same phenotypes.

CGP uses a simple mutation based $(1 + \lambda)$ evolutionary strategy as a search algorithm. The population size $1 + \lambda$ is often very small ($\lambda$ is typically between 1 and 15). The initial population is constructed either randomly or it can be seeded with a known solution as well (evolutionary optimization) [3]. In each generation, the best individual or a sibling with the same fitness value is passed to the next generation unmodified along with its $\lambda$ mutants. The mutation rate $m$ is usually set to modify up to 5 % randomly selected genes.

### 3.2 Fitness function

The principle of bent function synthesis by means of CGP is very similar to the case of combinational circuit design [2], since every Boolean function can be implemented by a combinational circuit. The difference lies in the fitness function. Unlike combinational circuits having fitness value equal to the total

number of wrong output bits, the fitness value of a bent function candidate is its nonlinearity, i.e. the lowest Hamming distance from all affine functions:

$$f(g) = \mathrm{NL}_g. \tag{1}$$

The fitness calculation is computationally very intensive, since the number of affine functions being compared with the candidate individual grows exponentially with the number of variables and the size of the truth table grows exponentially as well.

### 3.3 Efficient fitness evaluation

While evaluating an individual's fitness value, all active genes of the chromosome need to be traversed and their output values need to be calculated. The single output is then compared against all linear functions simply by `XOR`ing the values and counting the number of ones. There is no need to compare the values to the remaining affine functions (the complements of linear functions), since the following always holds true:

$$d(f, g) + d(f, g_\mathrm{c}) = 2^n, \tag{2}$$

where $f, g$ are arbitrary $n$-variable Boolean functions and $g_\mathrm{c}$ is complementary to $g$.

The entire evolutionary design process can be accelerated in the same way as it has been done in the case of combinational circuits [2]. The test vectors can be fed to the CGP individual in parallel, from 64 test vectors within a standard x86-64 register up to 256 or 512 test vectors using AVX extension or AVX-512 respectively.

In our previous work, the linear functions had to be precalculated and they had to reside in the memory [7]. However, the memory requirements of this approach are increasing rapidly with the number of input variables. The total memory requirements including the primary inputs, all linear functions and the CGP nodes are equal to $\frac{(n+2^n+n_\mathrm{r}\cdot n_\mathrm{c})\cdot 2^n}{8}$ bytes (see Table 4). While increasing the number of input variables, the memory portion allocated for the linear functions becomes major.

Significant memory savings can be achieved by introducing a new approach to evaluate the function's nonlinearity. Let $L_c = L_C$ be a $n$-variable linear Boolean function given uniquely by its code $c \in \{0, ..., 2^n - 1\}$, alternatively represented as a binary string $C = c_{n-1} \dots c_0 \in \{0, 1\}^n$. The function $L_c$ is then given by the equation:

$$L_c(x_{n-1}, ..., x_0) := x_{i_1} \oplus \cdots \oplus x_{i_m}, \tag{3}$$

where $i_j \in \{0, ..., n-1\}$ denote all positions in $C$, such as $c_{i_j} = 1$.

Note that all linear Boolean functions of $n$ variables have an unique code $c \in \{0, ..., 2^n - 1\}$ and vice versa, each $c \in \{0, ..., 2^n - 1\}$ is a valid code for an $n$-variable linear function. Thus, while evaluating the function's nonlinearity, all the codes $c \in \{0, ..., 2^n - 1\}$ have to be processed. The computed nonlinearity is

**Table 3.** Function $g$ nonlinearity computation with the use of Gray encoding.

| | gray code | function | XOR | result |
|---|---|---|---|---|
| thread 0 | $0000_2 = 0$ | $L_0 = 0$ | 0 | $g \oplus L_0$ |
| | $0001_2 = 1$ | $L_1 = x_0$ | $x_0$ | $g \oplus L_1$ |
| | $0011_2 = 3$ | $L_3 = x_1 \oplus x_0$ | $x_1$ | $g \oplus L_3$ |
| | $0010_2 = 2$ | $L_2 = x_1$ | $x_0$ | $g \oplus L_2$ |
| thread 1 | | | $f_2$ | $g \oplus L_2$ |
| | $0110_2 = 6$ | $L_6 = x_2 \oplus x_1$ | $x_2$ | $g \oplus L_6$ |
| | $0111_2 = 7$ | $L_7 = x_2 \oplus x_1 \oplus x_0$ | $x_0$ | $g \oplus L_7$ |
| | $0101_2 = 5$ | $L_5 = x_2 \oplus x_0$ | $x_1$ | $g \oplus L_5$ |
| | $0100_2 = 4$ | $L_4 = x_2$ | $x_0$ | $g \oplus L_4$ |
| thread 2 | | | $f_4$ | $g \oplus f_4$ |
| | $1100_2 = 12$ | $L_{12} = x_3 \oplus x_2$ | $x_3$ | $g \oplus L_{12}$ |
| | $1101_2 = 13$ | $L_{13} = x_3 \oplus x_2 \oplus x_0$ | $x_0$ | $g \oplus L_{13}$ |
| | $1111_2 = 15$ | $L_{15} = x_3 \oplus x_2 \oplus x_1 \oplus x_0$ | $x_1$ | $g \oplus L_{15}$ |
| | $1110_2 = 14$ | $L_{14} = x_3 \oplus x_2 \oplus x_1$ | $x_0$ | $g \oplus L_{14}$ |
| thread 3 | | | $f_{14}$ | $g \oplus L_{14}$ |
| | $1010_2 = 10$ | $L_{10} = x_3 \oplus x_1$ | $x_2$ | $g \oplus L_{10}$ |
| | $1011_2 = 11$ | $L_{11} = x_3 \oplus x_1 \oplus x_0$ | $x_0$ | $g \oplus L_{11}$ |
| | $1001_2 = 9$ | $L_9 = x_3 \oplus x_0$ | $x_1$ | $g \oplus L_9$ |
| | $1000_2 = 8$ | $L_8 = x_3$ | $x_0$ | $g \oplus L_8$ |

not dependent on the order, in which we proceed, but there is a permutation we can use with advantage. If we proceed in the order of the *Gray code*, each two successive linear functions differ in just one variable. The evaluation process is illustrated in Table 3. Starting with the function $L_{0000}$, the subsequent functions are $L_{0001}$, $L_{0011}$, $L_{0010}$, $L_{0110}$, $L_{0111}$, ...

By following the proposed procedure, one can compute the Hamming distances of the candidate function from all the linear functions (and their complements). There is no need to store the linear functions in memory, since all the `XOR` operations are performed with the candidate function's truth table and the input variables. This leads to a significant reduction of memory requirements, this approach needs only $\frac{(n+n_r \cdot n_c) \cdot 2^n}{8}$ bytes of memory. Even for a high number of input variables, the required memory portion fits to the cache[1] (see the comparison in Table 4), resulting in a performance increase. Moreover, the number of load operations is substantially reduced due to the sequential nature of the evaluative process.

The advantage of the sequential calculation is not noticeably eliminated even in the case of a parallel implementation. The linear functions can be uniformly divided between the threads, such that each thread processes the same number (except for indivisible cases) of successive linear functions (ordered with the Gray

[1] Considering a Xeon E5-2665 processor with 32 kB L1, 256 kB L2 and 20 MB shared L3 cache.

**Table 4.** Memory requirements (for 10 % active nodes in average).

| $n$ | nodes | active nodes | without optimization | | with optimization | |
|---|---|---|---|---|---|---|
| | | | memory | fits to | memory | fits to |
| 8 | $1 \times 100$ | 10 | 8.56 kB | L1 | 0.56 kB | L1 |
| 10 | $1 \times 150$ | 15 | 131.13 kB | L2 | 3.13 kB | L1 |
| 12 | $1 \times 200$ | 20 | 2.02 MB | L3 | 16.00 kB | L1 |
| 14 | $1 \times 250$ | 25 | 32.08 MB | M | 78.00 kB | L2 |
| 16 | $1 \times 300$ | 30 | 512.36 MB | M | 368.00 kB | L3 |
| 18 | $1 \times 350$ | 35 | 8.00 GB | M | 1.66 MB | L3 |

code). The only drawback of the parallel implementation lies in the initialization of the threads – each thread has to start with the last linear function of the preceding thread (but it is possible to precalculate it).

## 4 Intel Xeon Phi coprocessor

After a few decades of increasing the transistor density together with the operating frequency, the processor manufacturers had to come with a new approach to increase the performance due to the so called "Power Wall". While the single-threaded performance grew up rather slowly, the power consumption

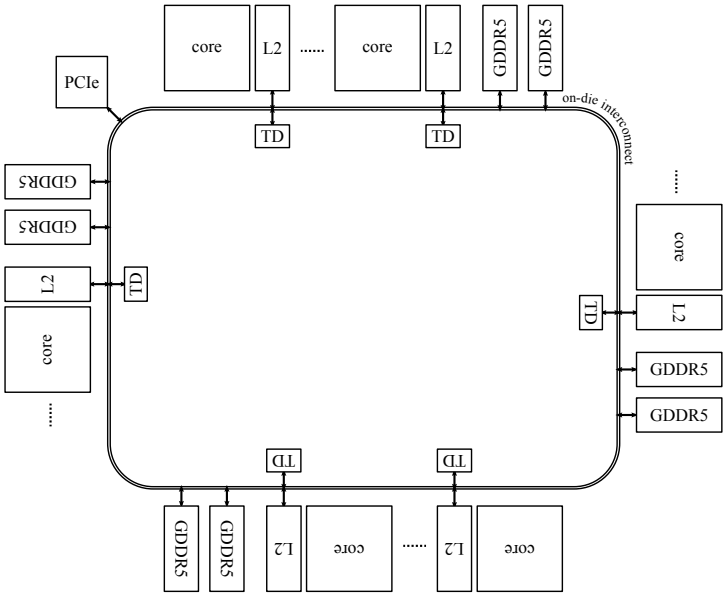



**Fig. 2.** Xeon Phi overall architecture.

of the last processors of the "era of higher processor speeds" exceeded reasonable limits. Since then, a greater emphasis was put to multi-core and power efficient processor architectures along with new parallel programming models and tools (libraries, compilers etc.) [14].

A continued rise in transistor density allowed to introduce even more parallelism; besides general-purpose computing on graphics processing units (GPGPU), a new *many-core* architectural concept emerged. Intel Xeon Phi coprocessor is an example of this approach, it has been designed for applications that can exploit vector instructions and are scalable enough to efficiently run in a huge number of threads [15]. Unlike GPGPU, the user can exploit standard programming model and thus reuse a lot of CPU optimized code. However, to reach the maximum performance, one has to seriously deal with manual optimizations. A deep knowledge of the microarchitecture is needed to achieve that.

The overall architecture of Xeon Phi coprocessor is depicted in Figure 2. The first generation code-named *Knights Corner* is made at a 22 nm design process, featuring 57–61 cores clocked at 1 GHz or more depending on the coprocessor model. The cores are interconnected by a high-speed bidirectional ring providing cache coherency across the entire coprocessor using a distributed Tag Directory (TD) mechanism. The communication over the on-die interconnect is transparent to the code allowing to employ the shared memory programming model. Up to 16 GDDR5 memory channels can be accessed over the ring [15].

The core's microarchitecture (see Figure 3) is based on the Intel Pentium P54c in-order superscalar architecture, significantly enhanced with the 64b support, 512b wide vector instructions (AVX-512), multithreading (up to 4 threads

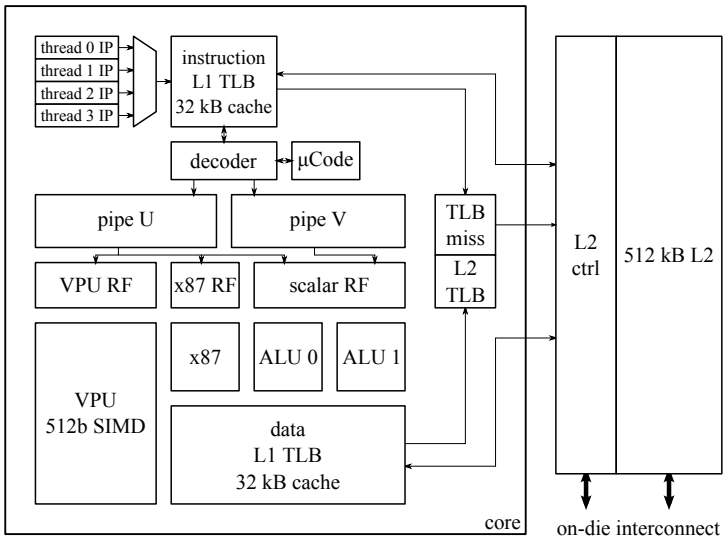



**Fig. 3.** Xeon Phi core architecture.

per core), power management and much more. Two instructions can be executed in parallel, one on the U-pipe and one on the V-pipe, if there is no conflict according to the pairing rules. The vector processing unit (VPU) is only available on the U-pipe. Scalar instructions have 1-cycle latency, while most vector instructions have 4-cycle latency with 1-cycle throughput. In order to fully utilize the execution units, at least two threads should be executed on each core due to a two-cycle latency of the instruction decoder.

The VPU is supplemented with a large register file containing 32 512b vector data registers and 8 16b vector mask registers for each HW thread separately. Each core includes 32kB L1 data and instruction caches and an inclusive 512kB L2 cache with a cache coherency protocol. The data caches don't implement any sophisticated prefetching mechanism, but there are special instructions for manual prefetching.

### 4.1 Optimizations

The evolutionary design of bent functions is based on computing the nonlinearities of candidate functions, which involves two main operations – `XOR` and `POPCNT` (population count, the number of ones). Despite the existence of a special `POPCNT` instruction, it can be implemented more efficiently on Xeon Phi. The principle is demonstrated in the Figure 4, the following listing shows a basic implementation [16]:

```
uint32_t popcnt(uint32_t x)
{
  x = (x & 0x55555555) + ((x >> 1) & 0x55555555);
  x = (x & 0x33333333) + ((x >> 2) & 0x33333333);
  x = (x & 0x0F0F0F0F) + ((x >> 4) & 0x0F0F0F0F);
  x = (x & 0x00FF00FF) + ((x >> 8) & 0x00FF00FF);
  x = (x & 0x0000FFFF) + ((x >> 16) & 0x0000FFFF);
  return x;
}
```

The algorithm is based on the divide and conquer strategy, in which the original problem (summing 32 bits) is divided into two subproblems (summing 16 bits); the subresults are then summed. This strategy is applied recursively, breaking the 16b fields into 8b, 4b, and so on [16]. This code can be further optimized by removing some unnecessary `AND` operations and shifts and vectorizing the code. The resulting implementation uses intrinsic AVX-512 instructions, 4×

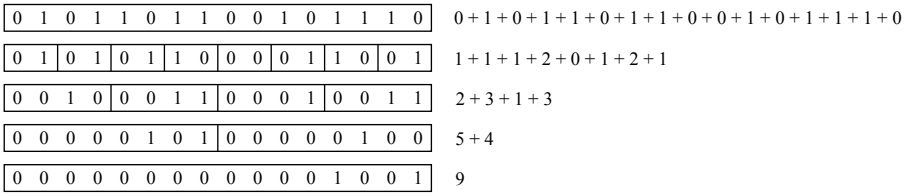



**Fig. 4.** Counting 1-bits using the divide and conquer strategy.

loop unrolling and memory prefetching. To compute the Hamming distance between two bit arrays, only 24 AVX, 8 load, 4 store and 8 prefetch instructions are needed for each 2048b block.

## 5 Experimental results

In this section, experiments regarding the performance (in terms of the evolution time) of the proposed approach to synthesize bent functions are presented. The CGP parameters were the same as for our previous experiments [7]: the functions set $\Gamma = \{\texttt{BUF}, \texttt{NOT}, \texttt{AND}, \texttt{OR}, \texttt{XOR}, \texttt{NAND}, \texttt{NOR}, \texttt{XNOR}\}$, population of 5 individuals, mutation rate 5 %. The number of rows was set to $n_\mathrm{r} = 1$ and the $l$-back parameter was maximal. No limitations on the number of generations were imposed, each run was successful. Each experiment was run 100× and the mean times were computed.

First, the comparison of the efficient fitness calculation approach (see Section 3.3) and the former implementation [7] is given in Table 5. For smaller number of variables (8–12), the former version is faster, but after exceeding the cache size (see Table 4), the new approach clearly outperforms the former implementation.

**Table 5.** Efficient fitness calculation analysis.

| $n$ | nodes $n_\mathrm{r} \times n_\mathrm{c}$ | mean time [s] | | speedup [-] |
|---|---|---|---|---|
| | | former | new | |
| 8 | $1 \times 100$ | 0.00470 | 0.00784 | 0.599 |
| 10 | $1 \times 150$ | 0.0602 | 0.1099 | 0.548 |
| 12 | $1 \times 200$ | 2.0443 | 2.1019 | 0.973 |
| 14 | $1 \times 250$ | 133.202 | 67.242 | 1.981 |
| 16 | $1 \times 300$ | 6223.66 | 1156.96 | 5.379 |

The performance of the Intel Xeon Phi 5110P (60 cores, 1.052 GHz) has been compared with a typical computer cluster node consisting of two Intel Xeon E5-2665 processors (8 cores, 2.4 GHz) and the impact of the manual optimizations according to Section 4.1 has been analysed. The evolutionary design utilized all available processor cores, i.e. 16 threads in the case of CPU and 240 threads

**Table 6.** Xeon Phi performance in terms of evolution time.

| $n$ | nodes $n_\mathrm{r} \times n_\mathrm{c}$ | mean time [s] | | | speedup [-] | |
|---|---|---|---|---|---|---|
| | | CPU | MIC | MIC opt. | MIC | MIC opt. |
| 12 | $1 \times 200$ | 0.73 | 6.75 | 0.84 | 0.11 | 0.87 |
| 14 | $1 \times 250$ | 9.51 | 53.17 | 3.62 | 0.18 | 2.63 |
| 16 | $1 \times 300$ | 109.62 | 122.03 | 40.88 | 0.89 | 2.68 |
| 18 | $1 \times 350$ | 2536.15 | 1764.32 | 814.02 | 1.44 | 3.12 |

in the case of Xeon Phi implementation. The achieved speedup of 3.12 for 18-variable bent functions is excellent considering the usual speedup of 2–3 reported by the literature [17]. The performance of the non-optimized version (the exactly same implementation as the CPU version) was very poor, clearly demonstrating the need for manual tuning.

## 6 Conclusions

Recently, a new approach to synthesize bent Boolean functions has been proposed [7]. However, the proposed implementation was not efficient enough for higher numbers of variables because of high memory requirements and computational demands.

Bent Boolean functions are of great importance for various cryptographic algorithms due to their properties – their nonlinearity makes cryptanalysis harder. The relative frequency of bent functions among all Boolean functions of the same arity is rapidly decreasing with the number of variables. Designing such functions is a computationally demanding task, but it has been shown that the evolutionary design can significantly outperform the state of the art methods [7].

In this paper, a new approach to speed up the evolutionary design of bent Boolean functions has been proposed. Besides the performance of the sequential implementation, which has been improved significantly for higher numbers of variables, the parallel efficiency of the new approach is substantially better allowing to exploit the many-core architecture of the Intel Xeon Phi coprocessor.

The performance of the Xeon Phi has been compared with a typical computer cluster node consisting of two Intel Xeon processors. Thanks to the same programming model (shared memory), the same implementation can be run on both targets. However, in order to reach the peak performance of the Xeon Phi, additional manual optimizations are needed. The simplified Knights Corner microarchitecture (especially the in-order character of the superscalar pipeline) is obviously a trade-off between the system complexity and user effort. Better code portability could be expected from the upcoming generation of Xeon Phi (Knights Landing) hopefully based on an out-of-order microarchitecture.

## 7 Acknowledgements

This work was supported by the Czech Science Foundation project 14-04197S. The author thanks the IT4Innovations Centre of Excellence for enabling these experiments.

## References

1. Miller, J., Thomson, P.: Cartesian genetic programming. In: Genetic Programming. Volume 1802 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2000) 121–132

2. Hrbacek, R., Sekanina, L.: Towards highly optimized cartesian genetic programming: From sequential via simd and thread to massive parallel implementation. In: GECCO '14 Proceedings of the 2014 conference on Genetic and evolutionary computation, Association for Computing Machinery (2014) 1015–1022
3. Vasicek, Z., Sekanina, L.: On area minimization of complex combinational circuits using cartesian genetic programming. In: 2012 IEEE World Congress on Computational Intelligence, Institute of Electrical and Electronics Engineers (2012) 2379–2386
4. Vasicek, Z., Bidlo, M.: Evolutionary design of robust noise-specific image filters. In: 2011 IEEE Congress on Evolutionary Computation, IEEE Computer Society (2011) 269–276
5. Hrbacek, R., Sikulova, M.: Coevolutionary cartesian genetic programming in fpga. In: Advances in Artificial Life, ECAL 2013, Proceedings of the Twelfth European Conference on the Synthesis and Simulation of Living Systems, MIT Press (2013) 431–438
6. Khan, G., Miller, J.: The cgp developmental network. In Miller, J.F., ed.: Cartesian Genetic Programming. Natural Computing Series. Springer Berlin Heidelberg (2011) 255–291
7. Hrbacek, R., Dvorak, V.: Bent function synthesis by means of cartesian genetic programming. In Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J., eds.: Parallel Problem Solving from Nature – PPSN XIII. Volume 8672 of Lecture Notes in Computer Science., Springer (2014) 414–423
8. Dobbertin, H.: Construction of bent functions and balanced boolean functions with high nonlinearity. In Preneel, B., ed.: Fast Software Encryption. Volume 1008 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1995) 61–74
9. Butler, J.T., Sasao, T.: Logic functions for cryptography - a tutorial. In: Proceedings of the Reed-Muller Workshop. (2009)
10. Shafer, J.L., Schneider, S.W., Butler, J.T., Stanica, P.: Enumeration of bent boolean functions by reconfigurable computer. In Sass, R., Tessier, R., eds.: FCCM, IEEE Computer Society (2010) 265–272
11. Rothaus, O.: On "bent" functions. Journal of Combinatorial Theory, Series A **20**(3) (1976) 300 – 305
12. Schneider, S.W.: Finding bent functions using genetic algorithms. Master's thesis, Naval Postgraduate School, Monterey (2009)
13. Miller, J.F., ed.: Cartesian Genetic Programming. Natural Computing Series. Springer Verlag (2011)
14. Hennessy, J., Patterson, D.: Computer Architecture: A Quantitative Approach. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science (2011)
15. Jeffers, J., Reinders, J.: Intel Xeon Phi coprocessor high-performance programming. Elsevier Waltham (Mass.), Amsterdam, Boston (Mass.), Heidelberg..., et al. (2013)
16. Warren, H.S.: Hacker's Delight. 2nd edn. Addison-Wesley Professional (2012)
17. Intel: Intel xeon phi product family performance. http://www.intel.com/content/www/us/en/benchmarks/xeon-phi-product-family-performance-brief.html (2013)

## A.4 Parallel Multi-Objective Evolutionary Design of Approximate Circuits

Radek Hrbacek

In *GECCO '15 Proceedings of the 2015 conference on Genetic and evolutionary computation*.
New York: Association for Computing Machinery, 2015.
ISBN 978-1-4503-3472-3, pp. 687-694.

# Parallel Multi-Objective Evolutionary Design of Approximate Circuits

Radek Hrbacek
Brno University of Technology, Czech republic
Faculty of Information Technology
ihrbacek@fit.vutbr.cz

## ABSTRACT

Evolutionary design of digital circuits has been well established in recent years. Besides correct functionality, the demands placed on current circuits include the area of the circuit and its power consumption. By relaxing the functionality requirement, one can obtain more efficient circuits in terms of the area or power consumption at the cost of an error introduced to the output of the circuit. As a result, a variety of trade-offs between error and efficiency can be found. In this paper, a multi-objective evolutionary algorithm for the design of approximate digital circuits is proposed. The scalability of the evolutionary design has been recently improved using parallel implementation of the fitness function and by employing spatially structured evolutionary algorithms. The proposed multi-objective approach uses Cartesian Genetic Programming for the circuit representation and a modified NSGA-II algorithm. Multiple isolated islands are evolving in parallel and the populations are periodically merged and new populations are distributed across the islands. The method is evaluated in the task of approximate arithmetical circuits design.

## Categories and Subject Descriptors

B.6.0 [**Hardware**]: Logic Design—*General*; I.2.8 [**Computing methodologies**]: Artificial intelligence—*Problem Solving, Control Methods, and Search*

## Keywords

Cartesian Genetic Programming; Parallel Evolutionary Algorithms; Multi-objective Optimization; Cluster; Combinational Circuit Design; Approximate Circuits

## 1. INTRODUCTION

While evolutionary design of digital circuits has been well established in the past, the correct functionality has always been an essential requirement put on the circuits. The other parameters, like the area, delay or power consumption, have been considered as secondary and have not been optimized as long as a fully working solution has been found. Recently, power efficiency has become the most important parameter of many real circuits. At the same time, a wide range of applications capable of tolerating imperfections (e.g. multimedia) has spread out. As a consequence, a new research field has been brought into being – the *approximate computing* [3].

The approximate digital circuits are designed in such a way that the functionality specification is not fully met in exchange for savings in terms of area, delay, power consumption etc. Although the circuit is not working properly, it can still be suitable for applications in which certain level of error is not recognizable (e.g. human perception and multimedia applications). Moreover, in some cases (e.g. low battery), the users could knowingly tolerate even more inaccuracy in order to extend the battery life.

After the first manual attempts to circuit approximation suffering from low scalability and efficiency [2, 7], a new class of systematic methods has been developed. The Systematic methodology for Automatic Logic Synthesis (SALSA) uses a quality function which decides whether a predefined quality constraint is met. The algorithm is allowed to modify the circuit as long as the quality constraint is not exceeded [19]. Another approach, Substitute-and-Simplify (SASIMI), looks for signal pairs having similar values with a high probability. By substituting one signal for the other, a part of the circuit can be removed resulting in area and power savings at the cost of an error introduced to the output [18].

The aforementioned methods have to be applied repeatedly with different error constraints if a set of trade-offs is demanded. In this paper, we propose a multi-objective evolutionary design approach which is capable of providing a whole set of trade-offs between a set of conflicting objectives. The proposed method is based on Cartesian Genetic Programming (CGP), widely used for the design of digital circuits, and a modified NSGA-II algorithm providing the multi-objective approach.

Since the evolutionary design is very computationally demanding [6], much emphasis has been put to the parallel implementation of the method. In order to make full use of a computer cluster, a spatially structured evolutionary algorithm has been introduced to the design process.

The proposed method has been evaluated in the task of approximate arithmetical circuits design with respect to three objectives – error, area and latency.


*GECCO'15,* July 11-15, 2015, Madrid, Spain.

http://dx.doi.org/10.1145/2739480.2754785

## 2. EVOLUTIONARY DESIGN OF DIGITAL CIRCUITS

In our previous work, we used Cartesian genetic programming to either design digital circuits from scratch [6] or to optimize existing circuits [14]. CGP, a branch of genetic programming, has been introduced by Miller [9]. While GP uses tree representation, an individual in CGP is represented by a directed acyclic graph of a fixed size. The candidate solution can have multiple outputs and intermediate results can be reused, which makes CGP very suitable for the design of digital circuits, e.g. arithmetic and logic circuits, digital filters, cryptography related Boolean functions, etc. [10, 5].

CGP uses a fixed-sized cartesian grid of $n_\mathrm{r} \times n_\mathrm{c}$ nodes interconnected by a feed-forward network (see Figure 1). Node inputs can be connected either to one of $n_\mathrm{i}$ primary inputs or to an output of a node in preceding $l$ columns. Each node has a fixed number of inputs $n_\mathrm{ni}$ (usually $n_\mathrm{ni} = 2$) and can perform one of the functions from the set $\Gamma$. Each of $n_\mathrm{o}$ primary circuit outputs can be connected either to a primary input or to a node's output. The area and delay of the circuit can be constrained by changing the grid size and the $l$-back parameter.

The genotype is of fixed length, whereas the phenotype is of variable length depending on the number of inactive nodes, i.e. nodes whose output is not used by any other node or primary output. This implies the existence of individuals with different genotypes but the same phenotypes, which is usually referred to as neutrality [20]. It was shown that for certain problems the neutrality significantly reduces the computational effort and helps to find more innovative solutions [8].

CGP uses a simple mutation based $(1 + \lambda)$ evolutionary strategy as a search mechanism. The population size $1 + \lambda$ is mostly very small, typically, $\lambda = 4$. The initial population is constructed either randomly (evolutionary design) or by mapping of a known solution to the CGP chromosome (evolutionary optimization). In each generation, the best individual is passed to the next generation unmodified along with its $\lambda$ offspring individuals created by means of point mutation operator. In case more individuals with the best fitness exist, a randomly selected one is chosen. The mutation rate $m$ is usually set to modify up to 5 % randomly selected genes.

In the case of digital circuit evolution, the fitness function usually corresponds to the quality of the candidate circuit measured as the number of correct output bits compared to a specified truth table (i.e. the Hamming distance). In order to obtain a fully working circuit, all combinations of input

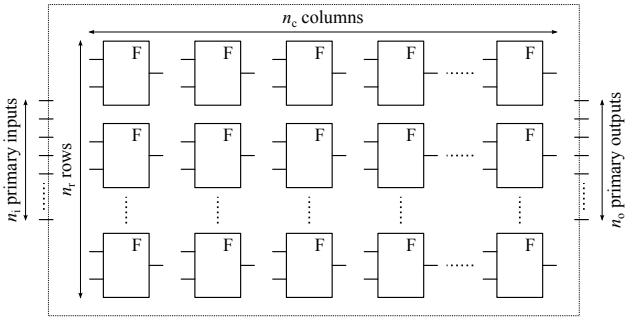



**Figure 1: Cartesian genetic programming scheme.**

values have to be evaluated. For a circuit with $n_\mathrm{i}$ inputs and $n_\mathrm{o}$ outputs, $2^{n_\mathrm{i}}$ test vectors need to be fetched to the primary inputs and $n_\mathrm{o} \cdot 2^{n_\mathrm{i}}$ output bits have to be verified so as to compute the fitness value.

The fitness calculation is computationally very intensive, since the number of test vectors grows exponentially with the number of primary inputs. Recently, it has been sped up by applying parallelism at various levels (data, thread, process) [6, 5] or by introducing formal methods, e.g. SAT solvers [14] or Binary Decision Diagrams (BDD) [16].

Besides the correct functionality, the demands placed on current digital circuits include the area of the circuit and its power consumption. The power consumption of digital circuits consists of two major components – the static and the dynamic power dissipation. The dynamic dissipation occurs while changing the state of the gates and thus it is highly dependent on the character of the gate's input signals. On the contrary, the static power consumption is rather constant and depends on the area of the circuit. The static dissipation has been substantially reduced by introducing the CMOS technology, which uses complementary connected transistors. However, with the decreasing size of the semiconductor technology process, the static dissipation is increasing due to rising leakage currents and is becoming the major component of the power consumption. Therefore, when evolving digital circuits with respect to the power consumption, the area of the circuit can be used to estimate the power consumption [13].

Another important characteristic of a digital circuit is the latency, i.e. the interval between the stimulation of the inputs and the response on the outputs. The latency can be determined by finding the longest path from the inputs to the outputs with respect to particular latencies of the gates along the path. Since the propagation delay of a gate differs for different transitions on the inputs, computing the total circuit latency would require simulating all possible input transitions on the whole circuit. The number of different transitions $N_\mathrm{t}$ grows rapidly with the number of primary inputs $n_\mathrm{i}$: $N_\mathrm{t} = 2^{n_\mathrm{i}} \cdot (2^{n_\mathrm{i}} - 1)$. Applying all these combinations is computationally intensive, therefore, an estimation must be used instead. In this paper, we assign each node function a fixed latency. When computing the overall circuit latency, the longest path from an input to an output considering the latencies of all gates along the path is considered.

### 2.1 Approximate circuits

Many computer systems or programs have the ability to tolerate some loss of accuracy or quality in the computational process and still produce meaningful and useful results. Significant area or energy-efficiency improvements can be achieved by relaxing the functionality requirement. For example, the growing popularity of portable multimedia devices offers a great scope for approximate computation, since human perception is limited and the users are ready to tolerate degraded quality of the multimedia content (e.g. video playback) in exchange for longer battery life. Automatic approximate computing techniques are being developed to speed up the design process and to find the trade-offs between the resources being shrunk (e.g. energy, time, area) and the inaccuracy of the computation.

Recently, several single-objective evolutionary approaches to design approximate circuits have been introduced [12]. Different error metrics have been utilized, starting with the

Hamming distance and introducing new metrics more suitable for arithmetical circuits, e.g. the worst case error, mean absolute error, relative error etc. [15]. In this paper, we use a compromise error metric which penalizes both the mean error and the isolated deviations – the mean squared error:

$$f_{\text{mse}} := \frac{\sum_{\forall i} \left( O^{(i)}_{\text{orig}} - O^{(i)}_{\text{approx}} \right)^2}{2^{n_{\text{i}}}}, \qquad (1)$$

where $O^{(i)}_{\text{orig}}$ is the decimal representation of the $i$-th circuit correct output and $O^{(i)}_{\text{approx}}$ is the individual's $i$-th output. The choice, which error metric to use, always depends on a concrete application.

Although several attempts to use multi-objective evolutionary algorithms exist [11], recent techniques have been based mainly on multi-phase single-objective approaches [15] or have used constrained resources in order to find approximate circuits with smaller area or power consumption [12, 17].

# 3. MULTI-OBJECTIVE CGP

Unlike the single-objective optimization, which enables to compare any two candidate solutions and decide which one is better, the multi-objective optimization leads to the existence of a whole range of trade-off solutions, if the objectives are conflicting. In the case of digital circuits design, the better the circuit works, the larger area and power consumption it has.

Many multi-objective evolutionary algorithms have been proposed, most of them are 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 principle can be seen in Figure 2, the Pareto optimal solutions are not dominated by any other solutions and form the so called *Pareto front*.

## 3.1 NSGA-II and its modifications

One of the most popular multi-objective evolutionary algorithms is the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [1]. It is based on sorting individuals 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 (see Algorithm 1) is very efficient, the overall complexity is $\mathcal{O}(MN^2)$, where $N$ is the population size and $M$ is the number of objectives.

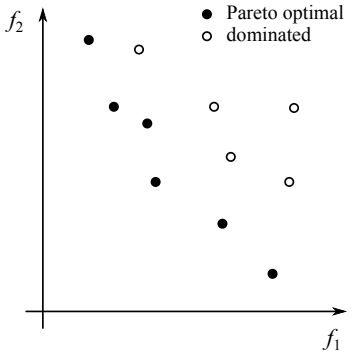



**Figure 2: Pareto optimal and dominated solutions.**

fast-non-dominated-sort($P$)
**foreach** $p \in P$ **do**
  $F_0 = \emptyset$
  $S_p = \emptyset$
  $n_p = 0$
  **foreach** $q \in P$ **do**
    **if** $p \prec q$ **then**
      $S_p = S_p \cup \{q\}$
    **end**
    **else if** $q \prec p$ **then**
      $n_p = n_p + 1$
    **end**
  **end**
  **if** $n_p = 0$ **then**
    $p_{\text{rank}} = 0$
    $F_0 = F_0 \cup \{p\}$
  **end**
**end**
$i = 0$
**while** $F_i \neq \emptyset$ **do**
  $Q = \emptyset$
  **foreach** $p \in F_i$ **do**
    **foreach** $q \in S_p$ **do**
      $n_q = n_q - 1$
      **if** $n_q = 0$ **then**
        $q_{\text{rank}} = i + 1$
        $Q = Q \cup \{q\}$
      **end**
    **end**
  **end**
  $i = i + 1$
  $F_i = Q$
**end**
$F = (F_0, F_1, \dots)$
**return** $F$

**Algorithm 1:** Fast non-dominated sort.

The solutions within the individual fronts are sorted according to the *crowding distance* metric, which helps to preserve a reasonable diversity along the fronts [1]. 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 (see Algorithm 2).

crowding-distance-assignment($P$)
$l = |I|$
**foreach** $p \in P$ **do**
  $p_{\text{dist}} = 0$
**end**
**foreach** *objective* $m$ **do**
  $P = \text{sort}(P, m)$
  $P[0]_{\text{dist}} = \infty$
  $P[l-1]_{\text{dist}} = \infty$ **for** $i$ *in* 1 *to* $l-2$ **do**
    $P[i]_{\text{dist}} = P[i]_{\text{dist}} +$
      $(P[i+1]_m - P[i-1]_m)/(f_m^{\max} - f_m^{\min})$
  **end**
**end**

**Algorithm 2:** Crowding distance assignment.

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.

Most real applications require to be able to constraint the solutions on particular objectives. NSGA-II offers a sim-

ple way to handle the constraints and keep the algorithm complexity low. Each solution can be either *feasible* or *infeasible*, the infeasible solutions are assigned a *constraint violation* according to the Algorithm 3. The constraints on the objective $m$ are denoted by $\langle c_m^{\min}, c_m^{\max} \rangle$.

```
constraint-violation-assignment(P)
foreach p ∈ P do
    p_constr_viol = 0
    foreach objective m do
        if p_m < c_m^min then
            p_constr_viol = p_constr_viol + (c_m^min − p_m)/f_m^max
        end
        if p_m > c_m^max then
            p_constr_viol = p_constr_viol + (p_m − c_m^max)/f_m^max
        end
    end
end
```

**Algorithm 3:** Constraint violation assignment.

When comparing two solutions, a feasible solution is always preferred. If both solutions are infeasible, the solution with smaller constraint violation is better. In the opposite case, when both solutions are feasible, the dominance depends on the rank and the crowding distance.

Since the original NSGA-II algorithm was based on a genetic algorithm, there must have been changes to use it with CGP [4, 11]. 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. The neutrality present in CGP causes a premature convergence, the Pareto fronts are flooded by individuals that are genotypically distinct but phenotypically identical. We propose to introduce a new *equivalence rank*, which enables to put the equivalent solutions in an order and preserve the neutrality character of the CGP. The principle can be seen from Algorithm 4. At the beginning, the population is randomly shuffled. Then, for each individual, the equivalence rank of all individuals (except for already processed ones) with the same fitnesses is incremented.

```
equivalence-rank-assignment(P)
foreach p ∈ P do
    p_eq_rank = 0
end
random_shuffle(P)
Q = P
foreach p ∈ P do
    Q = Q \ {p}
    foreach q ∈ Q do
        if p ≡ q then
            q_eq_rank = q_eq_rank + 1
        end
    end
end
```

**Algorithm 4:** Equivalence rank assignment.

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.

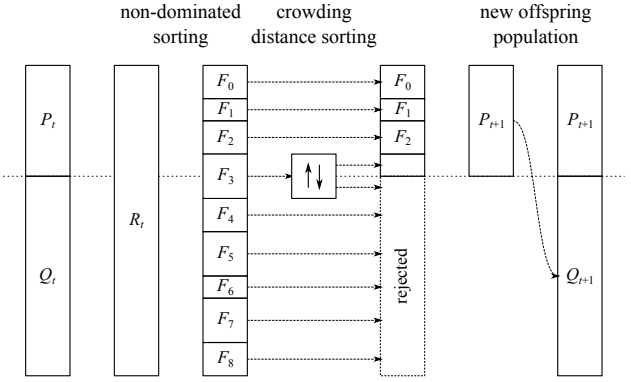



**Figure 3: NSGA-II algorithm scheme.**

Unlike the original NSGA-II algorithm, which uses a population of parents $P$ and an offspring population $Q$, both of size $N$, our modification enables us to set the offspring size independently. In each generation, $N_\mathrm{p}$ individuals are selected as parents and $N_\mathrm{q}$ offspring individuals are created by means of mutation. Besides the tournament selection, we use a new deterministic selection mechanism, which cyclically takes the individuals from the parent population $P$ and creates mutants.

The aforementioned principles make the multi-objective approach even more similar to the standard CGP. The overall algorithm works as follows:

```
nsga-ii(P_t, Q_t)
R_t = P_t ∪ Q_t
equivalence-rank-assignment(R_t)
constraint-violation-assignment(R_t)
F = fast-non-dominated-sort(R_t)
P_{t+1} = ∅
i = 0
while |P_{t+1}| + |F_i| ≤ N_p do
    crowding-distance-assignment(F_i)
    P_{t+1} = P_{t+1} ∪ F_i
    i = i + 1
end
crowding-distance-assignment(F_i)
sort(F_i, ≺_n)
P_{t+1} = P_{t+1} ∪ F_i [0 : (N_p − |P_{t+1}| − 1)]
Q_{t+1} = create-offspring(P_{t+1})
t = t + 1
```

**Algorithm 5:** Modified NSGA-II.

In each generation $t$, the populations $P_t$ and $Q_t$ form an unified population $R_t$. The individuals in $R_t$ are assigned the equivalence rank and the crowding distance. Then, the Pareto fronts are identified and the new parental population $P_{t+1}$ is filled with the individuals from the first fronts until $P_{t+1}$ is not overcrowded. The individuals from the last used Pareto front are sorted using the crowding distance and a fraction of them is selected just to fill the population $P_{t+1}$ (Figure 3).

# 4. PARALLEL MULTI-OBJECTIVE CGP

The evolutionary design is a very computationally demanding approach. In order to reduce the design time, one has to deal with a parallel implementation of the fitness function or search algorithm modifications. In our previous work, we have introduced parallelism at various levels (in-

struction, data, thread and process) to the CGP and sped up the design process significantly [6, 5]. However, the work was focused on single-objective design of (non-approximate) digital circuits having several specifics, e.g. small population size or the Hamming distance as the fitness function.

In the case of approximate arithmetical circuits, the fitness function (Equation 1) is much more computationally demanding and the implementation much less efficient. On the other hand, the population size of the multi-objective approach is much bigger, which makes the parallel processing of the individuals more efficient.

Besides the parallel implementation of the fitness evaluation, additional speed-up can be achieved by employing spatially structured evolutionary algorithms. Since CGP does not use any crossover operator, there is not a large scope of methods. However, a simple isolated islands model with a periodical exchange of the best individuals across the islands has been confirmed to be beneficial [6].

We propose to extend the multi-objective approach by introducing the isolated islands model. Unlike the single-objective case, the multi-objective algorithm requires to exchange the whole population. The Algorithm 6 is very similar to the single-population case, the only difference is that each $G_\mathrm{r}$ generations the populations are unified across the islands and a common Pareto front is identified on each island. Since the equivalence rank is assigned randomly to the individuals with the same fitness, the Pareto fronts on individual islands are phenotypically identical, but genotypically distinct. This principle should avoid the algorithm to converge prematurely and help to preserve the diversity across the populations.

nsga-ii-islands($P_t$, $Q_t$)
$R_t = P_t \cup Q_t$
**if** $t$ mod $G_\mathrm{r} = 0$ **then**
  $R_t$ = MPI_Allgather($R_t$)
**end**
equivalence-rank-assignment($R_t$)
constraint-violation-assignment($R_t$)
$F$ = fast-non-dominated-sort($R_t$)
$P_{t+1} = \emptyset$
$i = 0$
**while** $|P_{t+1}| + |F_i| \leq N_\mathrm{p}$ **do**
  crowding-distance-assignment($F_i$)
  $P_{t+1} = P_{t+1} \cup F_i$
  $i = i + 1$
**end**
crowding-distance-assignment($F_i$)
sort($F_i$, $\prec_\mathrm{n}$)
$P_{t+1} = P_{t+1} \cup F_i\,[0 : (N_\mathrm{p} - |P_{t+1}| - 1)]$
$Q_{t+1}$ = create-offspring($P_{t+1}$)
$t = t + 1$

**Algorithm 6:** NSGA-II with the isolated islands model.

# 5. EXPERIMENTAL RESULTS

In this section, experiments regarding the multi-objective design of arithmetical circuits are presented and the proposed modifications to the NSGA-II algorithm are examined. All experiments were performed on a computer cluster of 180 nodes with the following hardware configuration: 2× 8-core Intel E5-2665, 64 GB RAM, connected by Infiniband links. Each node was fully loaded with 16 threads, the evaluation of the population was parallelized using OpenMP.

The circuits were design with respect to 3 objectives – the mean squared error (as defined in Equation 1), the area of the circuit (approximate number of transistors considering common CMOS gates) and the latency (see Section 2). The CGP parameters were set similarly to the single-objective case [6], i.e. $\Gamma = \{\texttt{BUF}, \texttt{NOT}, \texttt{AND}, \texttt{OR}, \texttt{XOR}, \texttt{NAND}, \texttt{NOR}, \texttt{XNOR}\}$ and the mutation rate was set to 5 %, we used a linear CGP ($n_\mathrm{r} = 1$). The number of columns was $n_\mathrm{c} = 800$ in the case of the 4-bit multiplier and $n_\mathrm{c} = 100$ in the case of the adder. All experimental results were obtained by running 100 independent evolutionary runs.

## 5.1 Selection type

In Section 3.1, we have proposed a modification to the NSGA-II selection mechanism – a new deterministic selection. Furthermore, the offspring size $N_\mathrm{q}$ is not necessarily equal to the parental population size $N_\mathrm{p}$. In order to draw a comparison between the original tournament selection and the new deterministic selection, a number of experiments were carried out, the task was to design a combinational 4-bit multiplier.

The parental population size was set to $N_\mathrm{p} = 50$ and the offspring size was $N_\mathrm{q} \in \{50, 60, \ldots, 150\}$, the generation count was $G = 5000$. The same experiments were run for both tournament and deterministic selection.

The results can be seen in Figure 4. The quality of the resulting Pareto front was measured in terms of the Pareto front size (number of individuals) and the best error achieved. No matter how imperfect such comparison is, some trends can be inferred. The tournament selection is beneficial in the

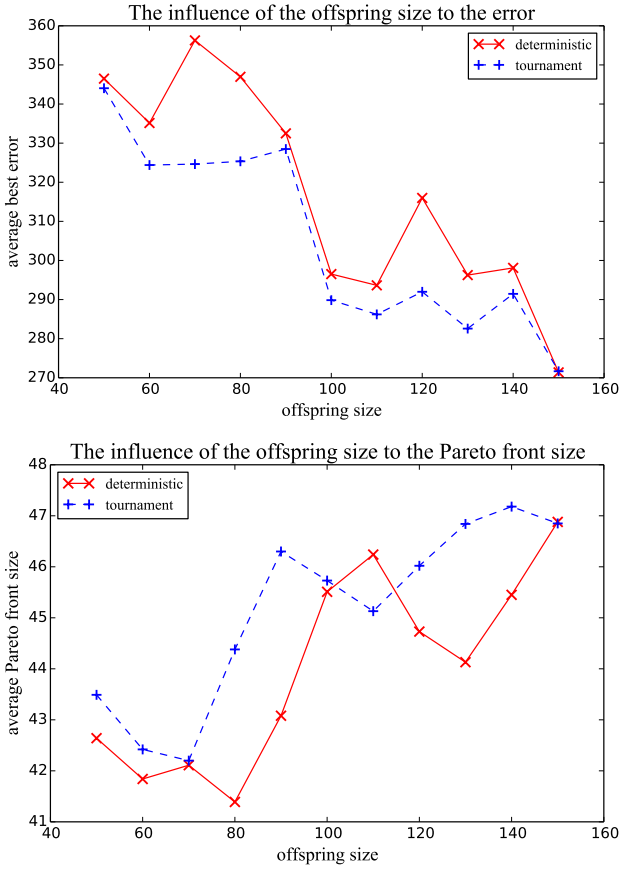
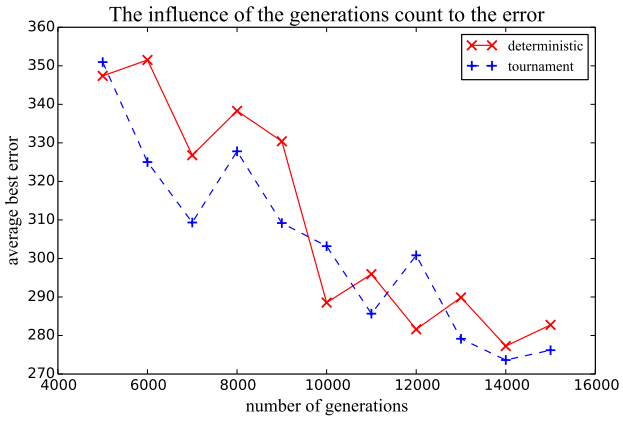



**Figure 4: The influence of the offspring size.**



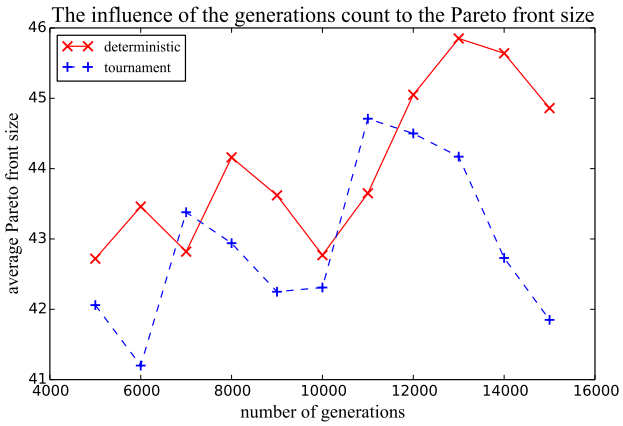




**Figure 5: The influence of the number of generations.**

cases, when $N_q$ is not an integer multiple of $N_p$. Otherwise, both selection types evince about the same performance.

Similarly to the single-objective case, one can adjust the number of generation and the population size so as the computational effort is the same. In our second experiment, the population size was fixed ($N_p = 50$, $N_q = 50$), but the number of generations was $G \in \{5000, 6000, \ldots, 15000\}$.

As can be seen in Figure 5, there is no conclusive difference between the two selection types. However, when comparing with the previous experiment (Figure 4), increasing the offspring size seems to be slightly more advantageous than increasing the number of generations.

## 5.2 Number of Islands

Recently, we have shown that a spatially structured evolutionary algorithm can speed up the evolutionary design of combinational circuits in comparison with a single-population process [6]. The Figure 6 shows the influence of the number of islands on the mean error achieved during the evolutionary design of a 4-bit multiplier. The single-population approach was compared to the multiple islands model with 2, 4 and 8 islands. The populations (each of $N_p = 100$ parental and $N_q = 100$ offspring individuals) were exchanged every $G_r = 500$ generations across the islands. It can be seen that increasing the number of islands significantly reduces the mean error during the whole evolutionary process. The more islands, the less generations is needed to achieve comparable results.

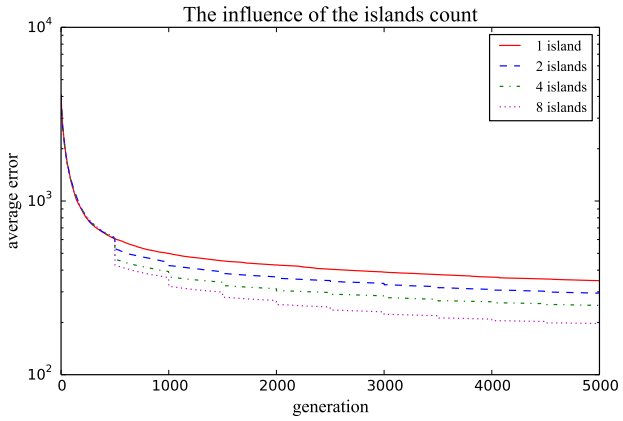



**Figure 6: The influence of the number of islands.**

## 5.3 Examples of Evolved Circuits

In this section, the proposed multi-objective approach is demonstrated on two examples of arithmetical circuits. Figure 8 shows the Pareto front of approximate 4-bit multipliers obtained after 1000000 generations. The population size was set to $N_p = 100$ parental and $N_q = 300$ offspring individuals and tournament selection was used to create the offspring population. Eight islands were exchanging the populations every $G_r = 1000$ generations. At the end of the evolution, 66 trade-off solutions were found having the mean squared error from 46.16 to 153.75, the area from 0 to 512 transistors and the latency from 0 to 19 gates. The extremely erroneous solutions are not shown in the Pareto front in Figure 8. Table 1 shows the output errors for all input combinations.

The same experimental setup was used for the design of combinational 4-bit adders. Since combinational adders are much less complex circuits than the multipliers, the number of generations was set to 100000. The resulting Pareto front can be seen in Figure 9, Table 2 shows the output errors for the individual with the lowest error. In comparison with the multiplier, the error is significantly smaller and the circuit has much smaller area (70 transistors) and latency (3 gates). The wiring diagram is depicted in Figure 7.

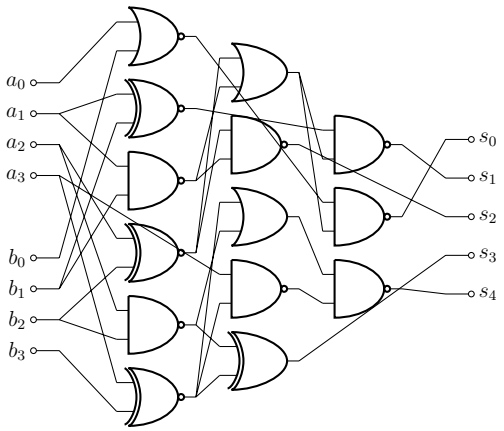



**Figure 7: The best 4-bit adder diagram.**

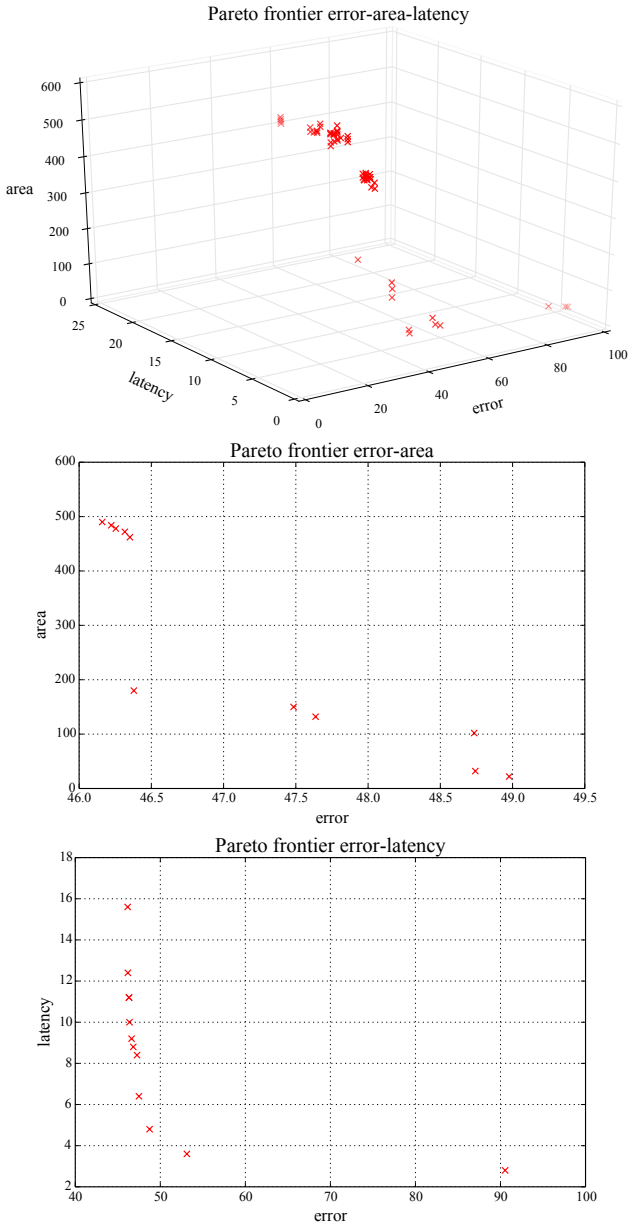



**Figure 8: Combinational 4-bit multiplier.**

| · | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 |
| 2 | 0 | 2 | 0 | 1 | 0 | 2 | 0 | 1 | 0 | 2 | 0 | 1 | 0 | 2 | 0 | 1 |
| 3 | 0 | 1 | 1 | -2 | 0 | 0 | -3 | -6 | 0 | 1 | 1 | -2 | -5 | -8 | -11 | -14 |
| 4 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | -1 | 0 | 0 | 0 | -2 | 0 | -4 | -1 | 2 | -3 | -8 | -5 | -2 | -7 | -12 |
| 6 | 0 | -1 | 0 | -3 | 0 | 1 | -5 | -11 | 0 | -1 | 0 | -3 | 8 | 2 | 4 | 1 |
| 7 | 0 | 0 | 1 | -6 | 0 | -4 | -11 | -18 | -1 | 0 | -7 | -14 | 0 | 1 | -3 | -10 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 1 | 0 | 1 | 0 | -1 | 2 | 0 | 0 | -3 | 0 | -4 | 3 | -6 | 1 | -8 |
| 10 | 0 | 2 | 0 | 1 | 0 | -3 | 3 | -7 | 0 | 3 | -5 | -15 | 0 | -3 | -13 | -23 |
| 11 | 0 | 3 | 0 | -2 | 0 | -8 | -3 | -14 | 0 | -4 | -15 | -26 | -5 | -16 | -27 | -38 |
| 12 | 0 | 0 | 0 | -5 | 0 | 0 | 8 | 4 | 0 | 0 | 0 | -5 | 16 | 4 | 0 | 1 |
| 13 | 0 | 1 | 0 | -8 | 0 | -2 | 2 | 1 | -1 | -6 | -3 | -16 | 4 | -1 | 2 | -4 |
| 14 | 0 | -1 | 3 | -11 | 0 | -7 | 0 | -3 | 0 | -1 | -13 | -27 | 8 | 1 | -5 | -19 |
| 15 | 0 | 0 | 1 | -14 | 0 | -12 | -3 | -10 | -1 | -8 | -23 | -38 | 0 | -4 | 14 | -1 |

**Table 1: Error table for the best multiplier.**

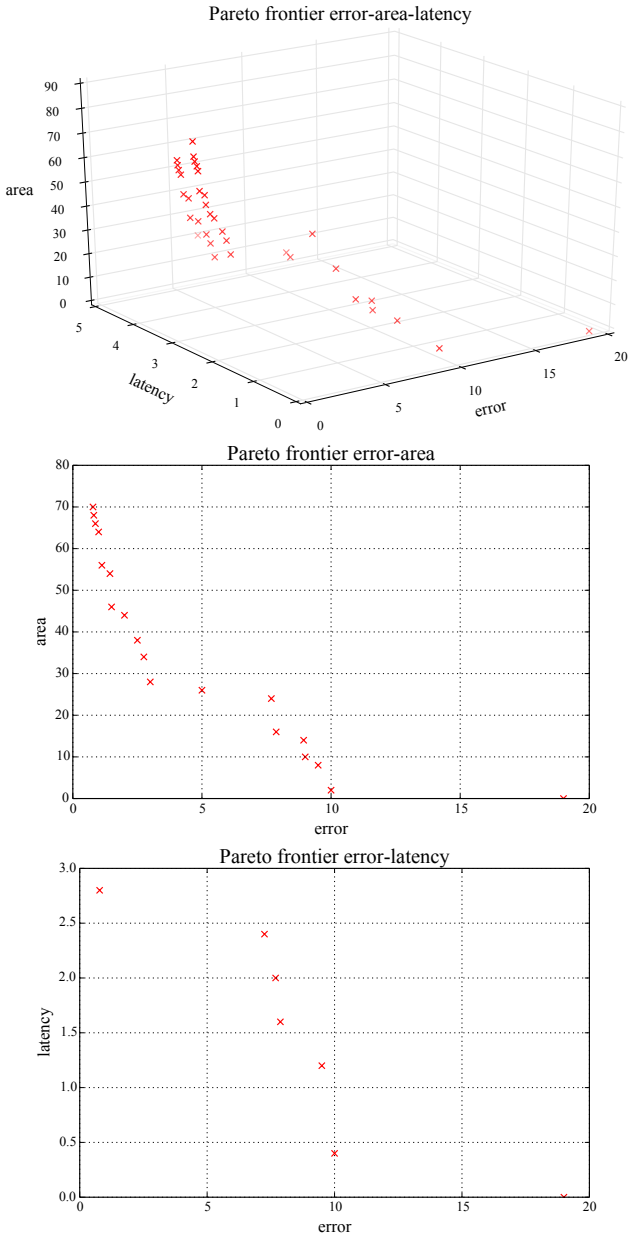



**Figure 9: Combinational 4-bit adder.**

| + | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | -2 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | -2 |
| 3 | 0 | -1 | 0 | -1 | 0 | -1 | -2 | -3 | 0 | -1 | 0 | -1 | 0 | -1 | -2 | -3 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 |
| 6 | 0 | 0 | -1 | -2 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | -2 | 0 | 0 | 0 | 0 |
| 7 | 0 | -1 | -2 | -3 | 0 | -1 | 0 | -1 | 0 | -1 | -2 | -3 | 0 | -1 | 0 | -1 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | -2 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | -2 |
| 11 | 0 | -1 | 0 | -1 | 0 | -1 | -2 | -3 | 0 | -1 | 0 | -1 | 0 | -1 | -2 | -3 |
| 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 |
| 14 | 0 | 0 | -1 | -2 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | -2 | 0 | 0 | 0 | 0 |
| 15 | 0 | -1 | -2 | -3 | 0 | -1 | 0 | -1 | 0 | -1 | -2 | -3 | 0 | -1 | 0 | -1 |

**Table 2: Error table for the best adder.**

# 6. CONCLUSIONS

Recently, a new application area for evolutionary algorithms has emerged, EAs have been confirmed to be competitive in the task of approximate circuits design [17]. The evolutionary design is a computationally demanding task, therefore, several approaches to speed up the entire process have been proposed [6].

In this paper, a new multi-objective evolutionary method for designing approximate digital circuits has been presented. The method is based on the well-known NSGA-II algorithm modified in order to be more suitable for the use with CGP. Besides a parallel implementation of the population fitness evaluation, a simple spatially structured algorithm is introduced for the purpose of speeding up the evolutionary process.

The proposed method has been evaluated in the task of approximate combinational multiplier and adder design. In comparison with existing methods, the multi-objective approach enables to obtain a set of Pareto optimal solutions in a single run.

In our future research, we will focus on increasing the scalability of the method in order to be able to design more complex circuits. For that purpose, the fitness function needs to be accelerated.

# 7. ACKNOWLEDGMENTS

This work was supported by the Czech Science Foundation project 14-04197S. The author thanks the IT4Innovations Centre of Excellence for enabling these experiments.

# 8. REFERENCES

[1] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii, Apr. 2002.

[2] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy. Low-power digital signal processing using approximate adders. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 32(1):124–137, Jan 2013.

[3] J. Han and M. Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *Test Symposium (ETS), 2013 18th IEEE European*, pages 1–6, May 2013.

[4] J. Hilder, J. Walker, and A. Tyrrell. Use of a multi-objective fitness function to improve cartesian genetic programming circuits. In *Adaptive Hardware and Systems (AHS), 2010 NASA/ESA Conference on*, pages 179–185, June 2010.

[5] R. Hrbacek. Bent functions synthesis on xeon phi coprocessor. In *Mathematical and Engineering Methods in Computer Science*, LNCS 8934, pages 88–99. Springer Verlag, 2014.

[6] R. Hrbacek and L. Sekanina. Towards highly optimized cartesian genetic programming: From sequential via simd and thread to massive parallel implementation. In *GECCO '14 Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 1015–1022. Association for Computing Machinery, 2014.

[7] P. Kulkarni, P. Gupta, and M. Ercegovac. Trading accuracy for power with an underdesigned multiplier architecture. In *VLSI Design (VLSI Design), 2011 24th International Conference on*, pages 346–351, Jan 2011.

[8] J. Miller and S. Smith. Redundancy and computational efficiency in cartesian genetic programming. *Evolutionary Computation, IEEE Transactions on*, 10(2):167–174, 2006.

[9] J. Miller and P. Thomson. Cartesian genetic programming. In *Genetic Programming*, volume 1802 of *Lecture Notes in Computer Science*, pages 121–132. Springer Berlin Heidelberg, 2000.

[10] J. F. Miller, editor. *Cartesian Genetic Programming*. Natural Computing Series. Springer Verlag, 2011.

[11] J. Petrlík and L. Sekanina. Multiobjective evolution of approximate multiple constant multipliers. In *IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems 2013*, pages 116–119. IEEE Computer Society, 2013.

[12] L. Sekanina and Z. Vasicek. Approximate circuits by means of evolvable hardware. In *2013 IEEE International Conference on Evolvable Systems (ICES)*, Proceedings of the 2013 IEEE Symposium Series on Computational Intelligence (SSCI), pages 21–28. IEEE Computer Society, 2013.

[13] L. Sekanina and Z. Vasicek. Evolutionary computing in approximate circuit design and optimization. In *1st Workshop on Approximate Computing (WAPCO 2015)*, pages 1–6, 2015.

[14] Z. Vasicek and L. Sekanina. Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. *Genetic Programming and Evolvable Machines*, 12(3):305–327, 2011.

[15] Z. Vasicek and L. Sekanina. Evolutionary design of approximate multipliers under different error metrics. In *17th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 135–140. IEEE Computer Society, 2014.

[16] Z. Vasicek and L. Sekanina. How to evolve complex combinational circuits from scratch? In *2014 IEEE International Conference on Evolvable Systems Proceedings*, pages 133–140. Institute of Electrical and Electronics Engineers, 2014.

[17] Z. Vasicek and L. Sekanina. Evolutionary approach to approximate digital circuits design. *IEEE Transactions on Evolutionary Computation*, 99(99):1–13, 2015.

[18] S. Venkataramani, K. Roy, and A. Raghunathan. Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 1367–1372, March 2013.

[19] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. Salsa: Systematic logic synthesis of approximate circuits. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 796–801, June 2012.

[20] T. Yu and J. Miller. Finding needles in haystacks is not hard with neutrality. In *Genetic Programming*, volume 2278 of *Lecture Notes in Computer Science*, pages 13–25. Springer Berlin Heidelberg, 2002.

## A.5 Automatic Design of Approximate Circuits by Means of Multi-Objective Evolutionary Algorithms

Radek Hrbacek, Vojtech Mrazek, Zdenek Vasicek

In *Proceedings of the 11th International Conference on Design & Technology of Integrated Systems in Nanoscale Era.*
Istanbul: Istanbul Sehir University, 2016.
ISBN 978-1-5090-0335-8, pp: 239-244.

2016 11th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)

# Automatic Design of Approximate Circuits by Means of Multi-Objective Evolutionary Algorithms

Radek Hrbacek
Brno University of Technology
Faculty of Information Technology
Czech Republic
ihrbacek@fit.vutbr.cz

Vojtech Mrazek
Brno University of Technology
Faculty of Information Technology
Czech Republic
imrazek@fit.vutbr.cz

Zdenek Vasicek
Brno University of Technology
Faculty of Information Technology
Czech Republic
vasicek@fit.vutbr.cz

***Abstract*—Recently, power efficiency has become the most important parameter of many real circuits. At the same time, a wide range of applications capable of tolerating imperfections has spread out especially in multimedia. Approximate computing, an emerging paradigm, takes advantage of relaxed functional requirements to make computer systems more efficient in terms of energy consumption, speed or complexity. As a result, a variety of trade-offs between error and efficiency can be found. In this paper, a design method based on a multi-objective evolutionary algorithm is proposed. For a given circuit, the method is able to produce a set of Pareto optimal solutions in terms of the error, power consumption and delay. The proposed design method uses Cartesian Genetic Programming for the circuit representation and a modified NSGA-II algorithm for design space exploration. The method is used to design Pareto optimal approximate versions of arithmetic circuits such as multipliers and adders.**

## I. Introduction

Approximate computing, an emerging paradigm, takes advantage of relaxed functional requirements to make computer systems more efficient in terms of energy consumption, computing speed or complexity. Error resilient applications can achieve significant savings while still serving their purpose with the same or a slightly degraded quality.

The complexity of computer systems is permanently growing and thus, automated design tools have to deal with more and more complex problems specified on higher level of abstraction than before. The same holds true for approximate computing. Even though new methods are emerging, there is a lack of methods for approximate computing offering a numerous set of trade-off solutions.

Evolutionary algorithms (EAs) have been confirmed to bring innovative solutions to complex problems. Recently, complex digital circuits have been optimized by means of EAs while the scalability of the methods has been improved substantially [5], [12]. Multi-objective EAs have been used to design simple approximate circuits from scratch [4].

In this paper, we propose an evolutionary based approach to design approximate circuits starting from a set of conventional fully working circuits. The method is evaluated in the task of approximate 8-bit adders and multipliers design.

## II. Approximate Computing

Recently, power efficiency has become the most important parameter of almost every computing platform. At the same time, a wide range of applications capable of tolerating imperfections in computations has spread out. As a consequence, a new research field – *approximate computing* – has emerged to investigate how computer systems can be made more efficient in terms of energy consumption, computing speed or complexity assuming that some errors are acceptable. It has been believed, that significant savings can be achieved by relaxing the requirement of perfect functionality thanks to the *error resilience* of some applications. Therefore, the *accuracy* of the system can be used as a design metric and inaccurate solutions can be accepted if an improvement in other parameters occurs.

The approximation can be introduced at various levels including the entire computer system architecture [6], particular components (e.g. ALU) [3], operating system, algorithm or even programming language [1]. As the complexity of today's computer systems grows, manual approximation is not an efficient design method. Hence, several automated approximate design methods have been introduced. The design of approximate circuits is typically based on modifying fully functional circuits.

The Systematic methodology for Automatic Logic Synthesis (SALSA) uses a quality function which decides whether a predefined quality constraint is met or not. The algorithm is allowed to modify the circuit as long as the quality constraint is not violated. SALSA has been applied to a number of problems, e.g. 32-bit adders, 8-bit multipliers, FIR filters, DCT blocks and others [18].

Another approach, Substitute-and-Simplify (SASIMI), looks for signal pairs having similar values with a high probability. By substituting one signal for the other, a part of the circuit can be removed resulting in area and power savings at the cost of an error introduced to the output. Moreover, SASIMI further extends the approach to synthesize quality configurable circuits, where at runtime, processing of selected input vectors is given an additional cycle to correct errors due to approximations [17].

Unlike the aforementioned methods, ABACUS (Automated

978-1-5090-0336-5/16/$31.00 ©2016 IEEE

Behavioral Approximate Circuit Synthesis) operates directly on the behavioral descriptions of circuits. ABACUS automatically generates approximate circuits from input behavioral descriptions by performing global transformations on an abstract synthesis tree (AST) created from the behavioral description. The outcome approximate circuits are still expressed in behavioral code and can be synthesized by means of standard synthesis tools. Complementary approximate computing methods, e.g. voltage over-scaling or manually created approximate components, may be still used [9].

Although most of the design methods deal with combinational circuits, there are methods capable of approximating sequential circuits. As an example, the Automatic Methodology for Sequential Logic Approximation (ASLAN) creates an approximate version of a sequential circuit that consumes lower energy, while meeting a specified quality constraint. ASLAN identifies combinational block in the sequential circuit that are amenable to approximation and iteratively approximates the entire sequential circuit using a gradient-descent approach [10].

## III. Evolutionary Design and Optimization

In our previous work, we used evolutionary algorithms to either design digital circuits from scratch [5] or to optimize existing circuits [12]. Recently, the evolutionary approach has been applied in the task of approximate circuits design with respect to multiple objectives [4].

### A. Cartesian Genetic Programming

The proposed method is based on CGP, in which a circuit is represented as a fixed-sized cartesian grid of $N_\mathrm{r} \times N_\mathrm{c}$ nodes interconnected by a feed-forward network (see Figure 1). Node inputs can be connected either to one of $N_\mathrm{i}$ primary inputs or to an output of a node in preceding $L$ columns. Each node has a fixed number of inputs $N_\mathrm{ni}$ and outputs $N_\mathrm{no}$ and can perform one of the functions from the set $\Gamma$. Each of $N_\mathrm{o}$ primary circuit outputs can be connected either to a primary input or to a node's output. The area and delay of the circuit can be constrained by changing the grid size and the $L$-back parameter.

The genotype is of fixed length, whereas the phenotype is of variable length depending on the number of inactive nodes, i.e. nodes whose output is not used by any other node or primary output (see Figure 2). This implies the existence of individuals with different genotypes but the same phenotypes, which is usually referred to as neutrality. It was shown that for certain problems the neutrality significantly reduces the computational effort and helps to find more innovative solutions [7].

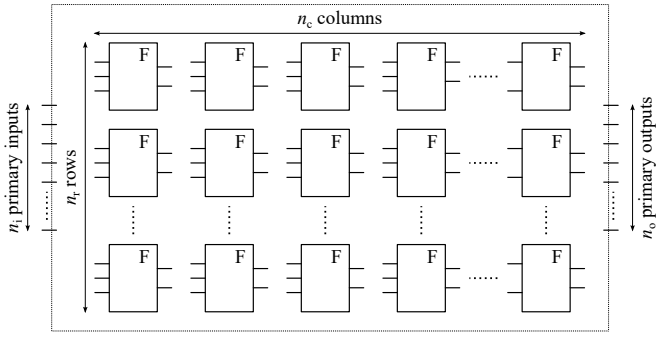



Fig. 1. Cartesian Genetic Programming.

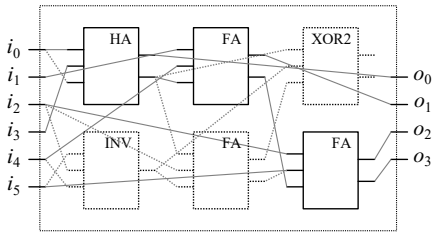



Fig. 2. Example of a CGP representation of 3-bit ripple-carry adder.

Standard (single-objective) CGP uses a simple mutation based $(1 + \lambda)$ evolutionary strategy as a search mechanism, the population size $1 + \lambda$ is mostly very small, typically, $\lambda = 4$. The initial population is constructed either randomly (evolutionary design) or by mapping of a known solution to the CGP chromosome (evolutionary optimization). In each generation, the best individual is passed to the next generation unmodified along with its $\lambda$ offspring individuals created by means of point mutation operator. In case more individuals with the best fitness exist, a randomly selected one is chosen. The mutation rate $m$ is usually set to modify up to 5 % randomly selected genes.

### B. Multi-Objective CGP

Unlike the single-objective optimization, which enables to compare any two candidate solutions and decide which one is better, the multi-objective optimization leads to the existence of a whole range of trade-off solutions, if the objectives are conflicting. In the case of digital circuits design, the better the circuit works, the larger area and power consumption it has.

Many multi-objective evolutionary algorithms have been proposed, most of them are 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

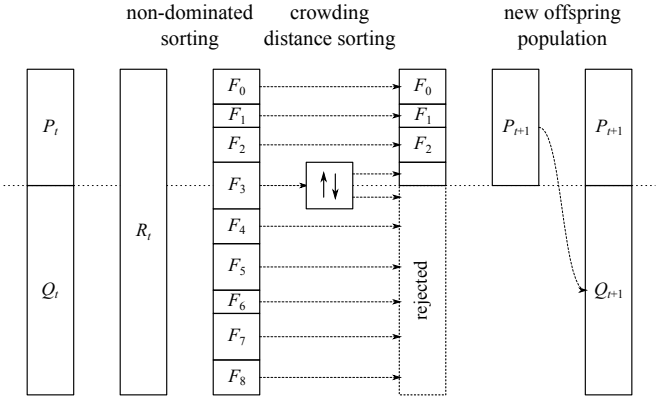



Fig. 3. Non-dominated Sorting Genetic Algorithm II.

not dominated by any other solutions and form the so called *Pareto front*.

One of the most popular multi-objective evolutionary algorithms is the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [2]. It is based on sorting individuals 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 principle of the algorithm can be seen in Figure 3. NSGA-II was recently applied to design approximate digital circuits from scratch, the convergence of the method was improved by using multiple islands [4]. The multi-objective approach was compared to the single-objective CGP in the task of approximate circuits design, however, the estimation of power consumption and delay of the circuits was rough [15].

### C. Function set

Since the goal of this paper is to optimize the circuits as much as possible, we use a subset of functions from a 180nm technology process library. 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 Table I.

| Function | Description | Area [$\mu m^2$] | Leakage power [nW] |
|---|---|---|---|
| BUF | Buffer (2x/4x) | 24/32 | 0.066/0.113 |
| INV | Inverter (1x/2x/4x/8x) | 16/16/24/40 | 0.022/0.036/0.073/0.147 |
| AND2 | 2-input AND (1x/2x) | 32/32 | 0.075/0.090 |
| OR2 | 2-input OR (1x/2x) | 32/32 | 0.075/0.090 |
| XOR2 | 2-input XOR (1x) | 56 | 0.161 |
| NAND2 | 2-input NAND (1x) | 24 | 0.039 |
| NOR2 | 2-input NOR (1x) | 24 | 0.035 |
| XNOR2 | 2-input XNOR (1x) | 56 | 0.161 |
| NAND3 | 3-input NAND (1x) | 36 | 0.056 |
| NOR3 | 3-input NOR (1x) | 64 | 0.055 |
| MUX2 | Multiplexor (1x) | 48 | 0.087 |
| AOI21 | 3-input AND/NOR (1x) | 32 | 0.052 |
| OAI21 | 3-input OR/NAND (1x) | 23 | 0.048 |
| FA | Full adder (1x) | 120 | 0.231 |
| HA | Half adder (1x) | 80 | 0.161 |

TABLE I
LIST OF USED FUNCTION CELLS.

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 is selected depending on the output load of the gate. The dynamic power and delay of the gates depend on the output load as well.

### D. Output Error

In the case of digital circuit evolution, the output error of the candidate circuit is often measured as the number of correct output bits compared to a specified truth table (i.e. the Hamming distance). In order to obtain a fully working circuit, $2^{N_\mathrm{i}}$ test vectors have to be evaluated so as to compute the fitness value. It can be sped up by applying parallelism at various levels [5] or by introducing formal methods, e.g. SAT solvers [12] or Binary Decision Diagrams (BDD) [14].

In the case of approximate circuits, Hamming distance is often not suitable. Instead, metrics based on the arithmetical distance, such as the worst case error, mean absolute error, relative error or others are usually used [13]. In this paper, we use the mean relative error:

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

where $O^{(i)}_{\mathrm{orig}}$ is the decimal representation of the $i$-th circuit correct output and $O^{(i)}_{\mathrm{approx}}$ is the individual's $i$-th output. In addition to that, we constrain the worst absolute and relative errors.

### E. Power Estimation

In order to estimate the power consumption of a candidate circuit, we propose to use a method based on the switching activity.

The power consumption of digital circuits can be divided into two different parts: dynamic and static power components. The first one occurs every time the output of a gate changes its logic value. In fact a low resistance path between the power rails is created during switching. Static power consumption is caused mainly by the leakage current which exists even when the circuit is in a stable state, i.e. not switching. Although the static power component has always been present, it has gained importance in sub-micrometer and nanometer devices [20].

Thus, the total power consumption has to be optimized by reducing static as well as dynamic part of the power consumption.

The power consumption $P = P_\mathrm{s} + P_\mathrm{d}$ of a candidate circuit is calculated as follows. Because the static part of the power consumption depends only on a function of a logic gate, the total static power consumption $P_\mathrm{s}$ can be obtained by summing static leakage for all gates of the candidate circuits. The leakage of each gate is defined by the technology specification file (so-called liberty file) for the target technology. The dynamic part $P_\mathrm{d}$ is defined as follows:

$$P_d = 0.5 \times C_{\mathrm{load}} \times V^2_{\mathrm{dd}} \times f \times E(\mathrm{transitions}), \qquad (2)$$

where $C_{\mathrm{load}}$ is the total load capacitance of the output (i.e. the sum of all input capacitances of the connected inputs defined in the liberty file), $V_{\mathrm{dd}}$ is the supply voltage, $f$ is target frequency and $E(\mathrm{transitions})$ is the expected value of the output transitions per global clock cycle (switching activity) [8].

We have used zero-delay model, i.e. glitches are not considered. Thus, the switching activity can be obtained using simulation of all input vectors, which is done during the

function verification. Total switching activity of a gate is calculated as follows:

$$E(\text{transitions}) = 2 \cdot (p_0 \cdot p_1) = 2 \cdot p_1 \cdot (1 - p_1), \quad (3)$$

where $p_0$ is probability that output of a considered gate is equal to logical zero, similarly $p_1$ is probability that the output is equal to logical one. There are more ways to determine transition probabilities. The simplest approach is to use the simulation and count the number of cases for which the output value was equal to 1.

### F. Propagation Delay Estimation

The delay of a candidate circuit is calculated using the parameters defined in the liberty timing file available for the utilized semiconductor technology. The delay $t_\mathrm{d}$ of a cell $c_i$ is modeled as a function of its input transition time $t_\mathrm{s}$ and capacitive load $C_\mathrm{l}$ 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:

$$\mathrm{Delay}(C) = \max_{\forall p \in \mathrm{path}} \sum_{c_i \in p} t_\mathrm{d}(c_i). \quad (4)$$

The capacitive load on the circuit outputs is chosen to be equal to the input capacitance of a buffer cell. The transition time on primary inputs corresponds to the transition time on the output of a buffer cell.

## IV. Experimental Results

In this section, experiments regarding the multi-objective design of arithmetical circuits are presented. The method was evaluated in the task of approximate 8-bit adders and multipliers design. The CGP parameters were set as follows: 500 individuals in the population, 5000000 generations, 10 islands, mutation rate 5 %, number of rows $N_\mathrm{r} = 1$. 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 3 objectives – the mean relative error (MRE), the power consumption of the circuit and the 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 are discarded.

### A. Initial population

In our previous research, we used random initial population to design simple digital circuits from scratch [5], [4]. For complex circuits, we seeded the initial population with a single known solution and optimized the circuit using CGP [12], [11], [15].

In this paper, we use a set of conventional circuits as the initial population. CGP chromosomes for 13 different adder and 6 different multiplier architectures were generated [19]. The power, area and delay estimates of those circuits can be found in Tables II, III. 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) architecures. The multipliers include Ripple-Carry Array, multiple Carry-Save Array and Wallace Tree architectures. All parameters in this section are related to the Ripple-Carry Adder and Ripple-Carry Array Multiplier architectures, since they are the most power efficient conventional architectures.

| Architecture | Power | Area | Delay |
|---|---|---|---|
| **Ripple-Carry Adder** | **100.00 %** | **100.00 %** | **100.00 %** |
| Carry-Select Adder | 201.18 % | 174.78 % | 61.15 % |
| Carry-Lookahead Adder | 414.74 % | 334.78 % | 61.99 % |
| HVTA (Brent-Kung) | 286.00 % | 201.74 % | 68.52 % |
| HVTA (Han-Carlson) | 286.00 % | 201.74 % | 68.52 % |
| HVTA (Kogge-Stone) | 371.48 % | 257.39 % | 59.77 % |
| HVTA (Sklansky) | 305.07 % | 215.65 % | 60.45 % |
| TA (Brent-Kung) | 282.99 % | 201.74 % | 67.25 % |
| TA (Han-Carlson) | 295.74 % | 212.17 % | 61.87 % |
| TA (Knowles) | 362.25 % | 257.39 % | 59.94 % |
| TA (Kogge-Stone) | 342.20 % | 243.48 % | 57.68 % |
| TA (Ladner-Fischer) | 282.99 % | 201.74 % | 67.25 % |
| TA (Sklansky) | 298.34 % | 212.17 % | 57.84 % |

TABLE II
POWER, DELAY AND AREA OF VARIOUS CONVENTIONAL 8-BIT ADDERS COMPARED TO RIPPLE-CARRY ADDER.

| Architecture | Power | Area | Delay |
|---|---|---|---|
| **Ripple-Carry Array** | **100.00 %** | **100.00 %** | **100.00 %** |
| Carry-Save Array using RCA | 102.30 % | 100.00 % | 71.16 % |
| Carry-Save Array using CSA | 108.42 % | 106.16 % | 62.03 % |
| Wallace Tree using RCA | 104.29 % | 107.39 % | 68.91 % |
| Wallace Tree using CLA | 116.10 % | 148.48 % | 51.26 % |
| Wallace Tree using CSA | 120.12 % | 122.35 % | 53.28 % |

TABLE III
POWER, DELAY AND AREA OF VARIOUS CONVENTIONAL 8-BIT MULTIPLIERS COMPARED TO RIPPLE-CARRY ARRAY MULTIPLIER.

### B. Results

Figure 4 shows 473 Pareto optimal 8-bit approximate adders evolved from the initial population of 13 conventional adders. Parameters of 9 selected evolved circuits can be found in Table IV. It can be seen that the Ripple-Carry Adder is optimal in terms of power consumption among the conventional architectures, but significant savings can be achieved when relaxing

| MRE | Power | Delay |
|---|---|---|
| 0.000 % | 244.78 % | 38.92 % |
| 0.135 % | 89.81 % | 79.93 % |
| 0.273 % | 85.99 % | 99.73 % |
| 0.396 % | 79.08 % | 96.29 % |
| 0.678 % | 71.89 % | 73.06 % |
| 0.942 % | 61.70 % | 59.59 % |
| 1.918 % | 47.66 % | 46.12 % |
| 2.939 % | 35.97 % | 33.92 % |
| 4.280 % | 33.39 % | 33.92 % |

TABLE IV
PARAMETERS OF EVOLVED APPROXIMATE 8-BIT ADDERS.

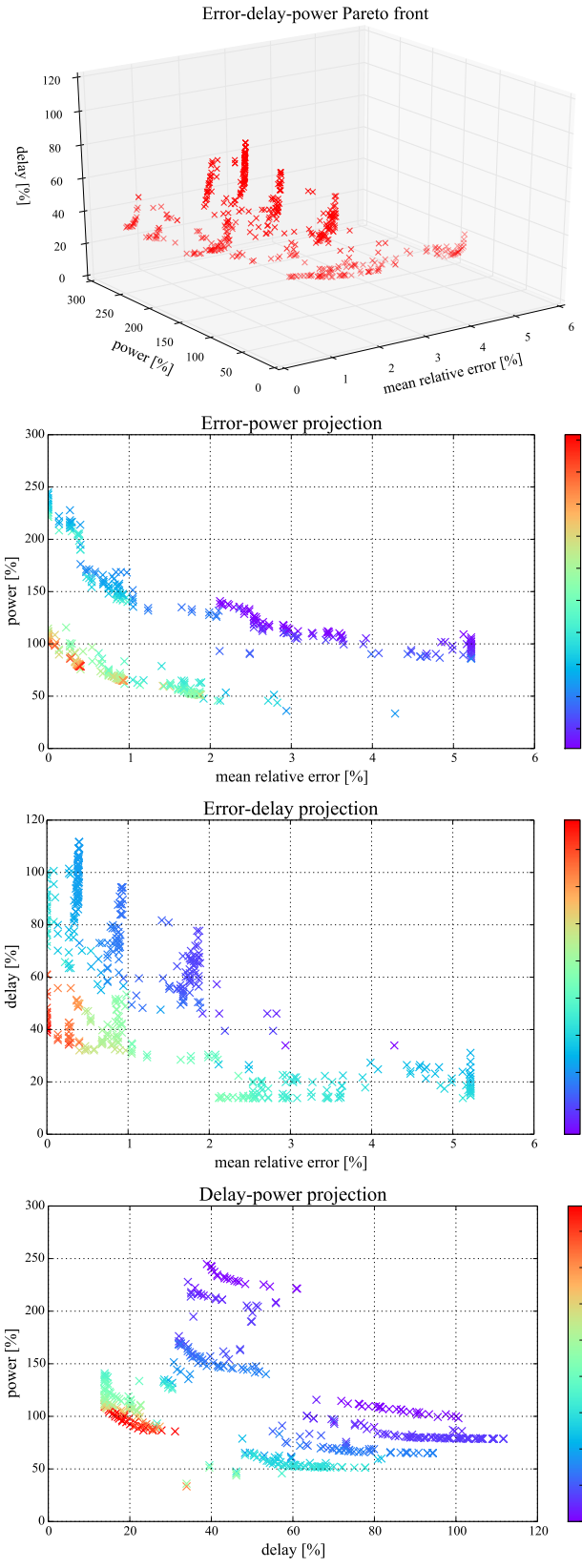



Fig. 4. Pareto front of evolved approximate 8-bit adders.

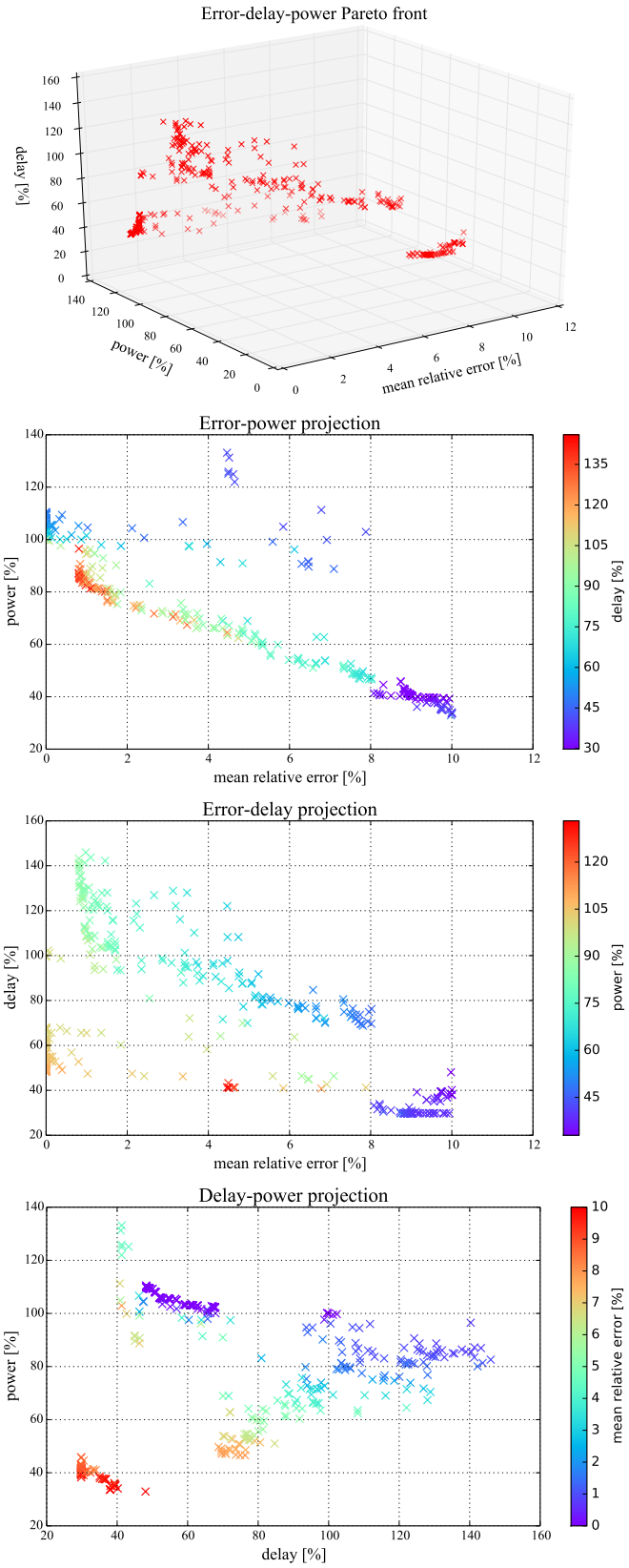



Fig. 5. Pareto front of evolved approximate 8-bit multipliers.

the requirement of perfect functionality. The delay of the Tree Adder with Sklansky architecture was overcome with multiple evolved circuits (at the cost of increasing the delay). The most efficient 8-bit adders have power consumption 33–36 % of the RCA with MRE of 3–4 %.

Similarly, Figure 5 shows 433 Pareto optimal 8-bit approximate multipliers that were evolved from 6 conventional circuits. Table V shows the parameters of 11 selected evolved multipliers. The Ripple-Carry Array Multiplier architecture was not overcome in terms of the power consumption when considering no error. The delay of Wallace Tree multipliers was improved to 48.23 % at the cost of a higher power consumption. The power savings are lower in comparison with the adders, for the same savings the error must be higher.

| MRE | Power | Delay |
|---|---|---|
| 0.000 % | 110.52 % | 48.23 % |
| 0.813 % | 88.70 % | 130.51 % |
| 0.951 % | 83.69 % | 113.02 % |
| 1.511 % | 76.15 % | 120.06 % |
| 3.092 % | 71.61 % | 96.79 % |
| 4.177 % | 66.19 % | 90.54 % |
| 5.334 % | 59.66 % | 80.60 % |
| 6.579 % | 51.01 % | 84.70 % |
| 8.218 % | 40.98 % | 33.94 % |
| 10.000 % | 33.74 % | 38.02 % |

TABLE V
PARAMETERS OF EVOLVED APPROXIMATE 8-BIT MULTIPLIERS.

## V. CONCLUSIONS

Recently, complex digital circuits were optimized by means of evolutionary algorithms [12]. Both single-objective and multi-objective approaches were applied to design approximate circuits from scratch [16], [4].

In this paper, the multi-objective approach was improved by seeding the initial population with a set of conventional fully working circuits instead of starting with a single conventional circuit or a random initial population. The method uses CGP for circuit representation and NSGA-II algorithm to handle multiple objectives.

The proposed method was evaluated in the task of approximate 8-bit adders and multipliers design. The circuits were designed with respect to three objectives – mean relative error, power consumption and delay. Contrasted to previous work, the method was able to evolve hundreds of Pareto optimal circuits with significant power consumption savings.

In our future research, we will focus on increasing the scalability of the method in order to design complex circuits. For that purpose, the use of formal methods will be investigated.

## ACKNOWLEDGMENTS

This work was supported by the Czech Science Foundation project 16-17538S, Brno University of Technology project FIT-S-14-2297 and the IT4Innovations excellence in science project (IT4I XS LQ1602).

## REFERENCES

[1] M. Carbin, S. Misailovic, and M. C. Rinard. Verifying quantitative reliability for programs that execute on unreliable hardware. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '13, pages 33–52, New York, NY, USA, 2013. ACM.

[2] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii, Apr. 2002.

[3] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy. Low-power digital signal processing using approximate adders. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 32(1):124–137, Jan 2013.

[4] R. Hrbacek. Parallel multi-objective evolutionary design of approximate circuits. In *GECCO '15 Proceedings of the 2015 conference on Genetic and evolutionary computation*, pages 687–694. Association for Computing Machinery, 2015.

[5] R. Hrbacek and L. Sekanina. Towards highly optimized cartesian genetic programming: From sequential via simd and thread to massive parallel implementation. In *GECCO '14 Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 1015–1022. Association for Computing Machinery, 2014.

[6] S.-L. Lu. Speeding up processing with approximation circuits. *Computer*, 37(3):67–73, Mar 2004.

[7] J. Miller and S. Smith. Redundancy and computational efficiency in cartesian genetic programming. *Evolutionary Computation, IEEE Transactions on*, 10(2):167–174, 2006.

[8] J. Monteiro, S. Devadas, A. Ghosh, K. Keutzer, and J. White. Estimation of average switching activity in combinational logic circuits using symbolic simulation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 16(1):121–127, Jan 1997.

[9] K. Nepal, Y. Li, R. Bahar, and S. Reda. Abacus: A technique for automated behavioral synthesis of approximate computing circuits. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6, March 2014.

[10] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan. Aslan: Synthesis of approximate sequential circuits. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6, March 2014.

[11] Z. Vasicek. Cartesian gp in optimization of combinational circuits with hundreds of inputs and thousands of gates. In *Genetic Programming, 18th European Conference, EuroGP 2015*, LCNS 9025, pages 139–150. Springer International Publishing, 2015.

[12] Z. Vasicek and L. Sekanina. Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. *Genetic Programming and Evolvable Machines*, 12(3):305–327, 2011.

[13] Z. Vasicek and L. Sekanina. Evolutionary design of approximate multipliers under different error metrics. In *17th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 135–140. IEEE Computer Society, 2014.

[14] Z. Vasicek and L. Sekanina. How to evolve complex combinational circuits from scratch? In *2014 IEEE International Conference on Evolvable Systems Proceedings*, pages 133–140. Institute of Electrical and Electronics Engineers, 2014.

[15] Z. Vasicek and L. Sekanina. Circuit approximation using single- and multi-objective cartesian gp. In *Genetic Programming*, LNCS 9025, pages 217–229. Springer International Publishing, 2015.

[16] Z. Vasicek and L. Sekanina. Evolutionary approach to approximate digital circuits design. *IEEE Transactions on Evolutionary Computation*, 99(99):1–13, 2015.

[17] S. Venkataramani, K. Roy, and A. Raghunathan. Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 1367–1372, March 2013.

[18] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. Salsa: Systematic logic synthesis of approximate circuits. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 796–801, June 2012.

[19] N. Weste and D. Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley Publishing Company, USA, 4th edition, 2010.

[20] A. Wiltgen, K. Escobar, A. Reis, and R. Ribas. Power consumption analysis in static cmos gates. In *Integrated Circuits and Systems Design (SBCCI), 2013 26th Symposium on*, pages 1–6, Sept 2013.

## A.6 Error Mitigation using Approximate Logic Circuits: A Comparison of Probabilistic and Evolutionary Approaches

Antonio José Sínchez-Clemente, Luis Entrena, Radek Hrbacek, Lukas Sekanina

*IEEE Transactions on Reliability.*
2016, vol. 65, no. 4.
ISSN 0018-9529, pp. 1871-1883.

# Error Mitigation using Approximate Logic Circuits: A Comparison of Probabilistic and Evolutionary Approaches

Antonio J. Sanchez-Clemente, Luis Entrena, Radek Hrbacek, Lukas Sekanina

***Abstract*—Technology scaling poses an increasing challenge to the reliability of digital circuits. Hardware redundancy solutions, such as Triple Modular Redundancy, produce very high area overhead, so partial redundancy is often used to reduce the overheads. Approximate logic circuits provide a general framework for optimized mitigation of errors arising from a broad class of failure mechanisms, including transient, intermittent and permanent failures. However, generating an optimal redundant logic circuit that is able to mask the faults with the highest probability while minimizing the area overheads is a challenging problem. In this work we propose and compare two new approaches to generate approximate logic circuits to be used in a TMR schema. The probabilistic approach approximates a circuit in a greedy manner based on a probabilistic estimation of the error. The evolutionary approach can provide radically different solutions that are hard to reach by other methods. By combining these two approaches, the solution space can be explored in depth. Experimental results demonstrate that the evolutionary approach can produce better solutions, but the probabilistic approach is close. On the other hand, these approaches provide much better scalability than other existing partial redundancy techniques.**

***Index Terms*—Approximate logic circuit, error mitigation, evolutionary computing, Single-Event Transient, Single-Event Upset.**

## ACRONYMS AND ABBREVIATIONS

| | |
|---|---|
| CGP | Cartesian Genetic Programming |
| DWC | Duplication With Comparison |
| EDAC | Error Detection And Correction |
| MA | Mandatory Assignment |
| SAT | Satisfiability |
| SMA | Set of Mandatory Assignments |
| SET | Single-Event Transient |
| SEU | Single-Event Upset |
| TMR | Triple Modular Redundancy |

## NOTATION

| | |
|---|---|
| $EP$ | Total Error Probability |
| $EA$ | Estimated Area |
| $P(f)$ | Probability of testing a fault $f$ |
| $G$ | Original circuit with no approximation |
| $H$ | Over-approximate circuit |
| $F$ | Under-approximate circuit |

This work was supported by the Ministry of Economy and Competitiveness of Spain under project ESP2015-68245-C4-1-P, and by the Czech science foundation project GA14-04197S and 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.

A. J. Sanchez-Clemente and L. Entrena are with the Electronic Technology Department, Universidad Carlos III de Madrid, Madrid 28911, Spain (e-mail: ajscleme@ing.uc3m.es; entrena@ing.uc3m.es).

R. Hrbacek and L. Sekanina are with the Faculty of Information Technology, IT4Innovations Centre of Excellence, Brno University of Technology, Brno 61266, Czech Republic (e-mail: ihrbacek@fit.vutbr.cz; sekanina@fit.vutbr.cz)

## I. INTRODUCTION

The impact of transient, intermittent and permanent failures on digital circuits is steadily increasing with technology scaling. Circuits manufactured on advanced technologies are more prone to errors due to several reasons, which are primarily related to the shrinking of transistor dimensions and the increase in the total number of gates per chip [1].

On the one hand, manufacturing process variations are a dominant source of static variability which may significantly affect yield. As a consequence, it is now common practice to use error correction codes or hardware redundancy in circuits with a regular internal structure, such as memories or programmable logic devices (FPGAs). In the past, variations were mostly due to imperfect process control, but now intrinsic atomistic effects, such as Random Dopant Fluctuations (RDF) or Line Edge Roughness (LER) have become relevant in sub-45-nm technologies, as devices of atomic sizes are achieved [2]. Due to the increasing difficulty of testing, some defects may escape manufacturing test and may cause intermittent failures resulting in errors during normal operation. Furthermore, transistor aging effects, such as negative-bias temperature instability (NBTI), also increase intermittent gate failures during the lifetime of a chip. Manufacturing variations, supply voltage variations, temperature variations and aging-related effects in digital circuits pose an increasing challenge to reliability [3].

Radiation-induced soft errors caused by ionizing particles, mainly neutrons at the atmospheric level and other particles in space environments have also become a big concern. In the past, transient effects in memory elements, known as Single-Event Upsets (SEUs), were the primary concern. However, for advanced technologies SEU protection is not enough, as transient effects in combinational logic gates, known as Single-Event Transients (SETs), are becoming very relevant [4]. Protection against SETs is much more difficult to achieve and typically involves a large amount of redundancy. Finally, post-silicon technologies such as carbon nanotubes are intrinsically less robust and require fault-tolerance [5].

Hardware redundancy is often used in safety- and mission-critical applications to mitigate the effects of transient errors,

permanent errors or configuration errors in FPGAs. Duplication With Comparison (DWC) or Triple Modular Redundancy (TMR) are well-known examples of techniques that provide concurrent error detection and correction capabilities, respectively. However, these techniques typically introduce very large overheads, which is more than 200 % in the case of TMR. When such overhead is not acceptable, partial redundancy is used in order to find a good balance between the reliability requirements and the area, power and performance requirements [6].

Approximate logic circuits provide a general framework for optimized mitigation of errors arising from a broad class of failure mechanisms, including transient, intermittent and permanent failures. An approximate logic circuit is a circuit that performs a possibly different but closely related logic function to the original circuit. As it is not required to exactly match the original circuit, the approximate circuit can be smaller but it can still be used to detect or correct errors where it overlaps with the original circuit. Approximate logic circuits can be used in TMR instead of exact copies of the original design and the designer can select the level of approximation. A closer approximation provides higher fault tolerance but also increases the area and power. In contrast, this continuous trade-off is not possible when exact TMR is used. However, generating an optimal redundant logic circuit that is able to mask the faults with the highest probability while minimizing the area and power overheads is a challenging problem.

In this work we propose and compare two new approaches to generate approximate logic circuits to be used in TMR. First of all, a probabilistic approach is proposed which is based on dynamic probability estimations. This approach takes advantage of strongly coupling the approximation method and the error estimation method by using stuck-at faults. Departing from the original target circuit, approximate logic circuits are built by iteratively forcing original circuit lines to constant values. The reduction in error mitigation that is produced by this type of transformations can be related to the probability of detecting associated stuck-at faults, which is the metric commonly used to estimate the error coverage. This approach can be used in a greedy manner to remove the logic with the lowest probability of producing an error while the required reliability target is met. However, it is well-known that greedy algorithms may often produce suboptimal results because they may get stuck in local minima. The second approach we propose in this work is based on evolutionary algorithms. Evolutionary algorithms are used in many applications to solve hard optimization and design problems. As the method is intrinsically based on the trial and error approach, it is usually very time consuming, but, on the other hand, capable of discovering solutions hard to reach by other methods. One of the major advantages of evolutionary algorithms is the ability to get out from local minima and increase the chances to reach global minima. Thus, evolutionary algorithms can provide radically different solutions. A comparison with the probabilistic approach is carried out in this work in order to contrast their respective capabilities.

This paper is organized as follows. Section II summarizes previous work and introduces approximate logic circuits and evolutionary circuit design. Section III describes the probabilistic approach. Section IV deals with the evolutionary approach. Experimental results are presented in Section V. Conclusions are given in Section VI.

## II. Previous work

Fault-tolerance techniques are classically classified into hardware redundancy, information redundancy and time redundancy techniques [7]. Among the hardware redundancy techniques, Triple Modular Redundancy (TMR) is a well-known error masking technique that is widely used in critical applications. TMR can be used at different levels of abstraction, from system to transistor level, and can protect against transient and permanent errors. Information redundancy techniques, such as Error Detection and Correction (EDAC) codes, can be very effective for single or double errors. Thus, EDAC codes are typically applied to memories or communication protocols, but they cannot be used in the general case because a low multiplicity of errors cannot be guaranteed. Finally, time redundancy is intrinsically non-robust to permanent failures and may introduce severe performance penalties.

The capability of TMR to mitigate both transient and permanent errors makes it a good technique to tackle the variety of potential failure mechanisms that must be considered for advanced technologies. However, TMR suffers from high overhead in terms of area and power (more than 200 %). To alleviate this overhead, alternative techniques have been proposed based on partial error masking. Without loss of generality, we will focus on combinational circuits. The extension to sequential circuits is trivial by applying TMR to the sequential elements along with the combinational elements.

An early partial error masking approach is proposed in [8] which consists on triplicating and voting the nodes with the highest soft error susceptibility. Subsequent approaches attempt to insert redundancies that protect against the most common errors or to resynthesize the circuit to improve reliability. In [9], an approach is proposed to provide protection for the most common output combinations. In [10], the authors propose the use of implications to build redundant logic that checks for violation of behavioural constraints.

In a recent work [11], small sub-circuits have been extracted and resynthesized using two-level techniques and fast extraction algorithm. The resynthesized circuits have been then merged to produce the final fault-tolerant circuit. Combinational restructuring has been used in [12] to improve the masking properties of a circuit. This approach takes advantage of conditions already present in the circuit, such as observability don't-cares. Other approaches use wire addition and removal for combinational restructuring [13]. Finally, there are approaches that use approximate logic circuits [14] for partial error detection and masking. These approaches are reviewed in the following section.

### A. Approximate logic circuits

The concept of approximate logic circuit or function provides a systematic framework for the implementation of fault-tolerant combinational logic circuits. Given a logic function $G$,

an approximate logic function is a function $\hat{G}$ that performs a possibly different but closely related function. The approximation divides the input space into two subspaces: the subset of input vectors for which $G$ and $\hat{G}$ produce the same output (correct subspace) and the subset of input vectors for which $G$ and $\hat{G}$ produce different outputs (incorrect subspace). The quality of an approximation is evaluated as the relative size of the correct subspace.

Approximations can be classified as unidirectional or bidirectional [15]. An approximation is called unidirectional if the incorrect subspace is either a subset of the on-set or a subset of the off-set of $G$. In the first case, $\hat{G}$ is called an under-approximation or on-set unidirectional approximation of $G$. Similarly, in the second case $\hat{G}$ is called an over-approximation or off-set unidirectional approximation of $G$. In the sequel, the under-approximations and over-approximations of a function $G$ will be denoted respectively as $F$ and $H$.

By definition, a unidirectional approximation satisfies an implication relationship. If $F$ is an under-approximation of $G$, then $F = 1 \Rightarrow G = 1$ and, conversely, $G = 0 \Rightarrow F = 0$. The incorrect subspace corresponds to the input vectors that produce $G = 1$ and $F = 0$, i.e., all input vectors in the incorrect subspace produce unidirectional $1 \rightarrow 0$ errors. If $H$ is an over-approximation of $G$, then $H = 0 \Rightarrow G = 0$ and, conversely, $G = 1 \Rightarrow H = 1$. In this case, the incorrect subspace corresponds to the input vectors that produce $G = 0$ and $H = 1$, i.e., all input vectors in the incorrect subspace produce unidirectional $0 \rightarrow 1$ errors. Bidirectional approximations do not satisfy an implication relationship and can produce both $0 \rightarrow 1$ and $1 \rightarrow 0$ errors.

Partial logic masking can be obtained by using a TMR schema in which two of the copies are replaced by approximate logic circuits, as shown in Fig. 1. Note that the approximate logic circuits may produce incorrect outputs even in the absence of faults. To ensure these incorrect outputs are masked, it is required that the incorrect subspaces of the two approximations do not overlap, so that at most one of the circuits is allowed to produce an incorrect output for any input vector. This condition is met by using an under-approximation $F$ and an over-approximation $H$ in the TMR schema.

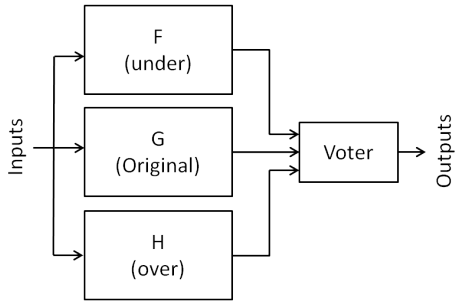



Fig. 1. Error masking schema using approximate logic circuits

The error masking capabilities of this schema can be better explained using the diagram shown in Fig. 2 [14], where the on-sets of the original and the approximate logic functions are represented. In the areas where the original and the approximate logic functions overlap (correct subspace), all circuits produce the same output value. Because the three circuits are implemented separately, a single fault can only affect one of them at a time and its effect will be masked. In the areas where the approximate functions do not overlap (incorrect subspace), one of the approximate functions produces an incorrect result. This incorrect result is masked, but a fault in any of the other two circuits may cause an additional incorrect result which cannot be masked by the majority voter. Therefore, the probability of error is directly related to the probability of faults that can propagate errors to the outputs for input vectors in the incorrect subspace of any of the approximate circuits. The goal is to find approximate circuits that minimize this probability and can be implemented with a reduced amount of logic.

It must be noted that approximate logic circuits are susceptible to errors that may not be masked. This may happen in the incorrect subspace, if a fault in an approximate circuit causes the two approximate circuits to agree on an incorrect result. However, this situation is detectable, because it is impossible by construction that the two approximate circuits disagree with the original circuit unless there is a fault. Thus, all errors produced in the approximate logic circuits are either masked or detectable. Generally, the contribution of the approximate logic circuits to the error rate is compensated by the error masking on the original circuit. However, if this is not the case, the voter can be complemented with an error detector. This way, it is guaranteed that the failure probability always reduces as the quality of the approximation increases.

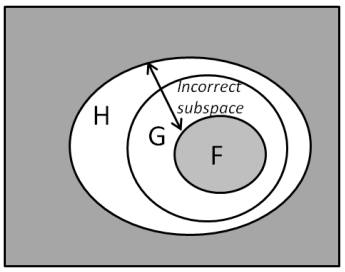



Fig. 2. Graphical representation of the relationship among the original and the approximate functions

An algorithm for technology-independent synthesis of approximate logic functions is proposed in [16]. This algorithm utilizes technology-independent networks and tries to approximate the local logic function of a node by moving minterms from its off-set or on-set into don't-cares. Minterm selection is based on the logic function of the node or, alternatively, local observability don't-cares can be used to expand the space from which minterms can be selected. However, as this approach may lead to an incorrect approximation, a SAT solver is used to ensure correctness. This approach is extended in [15] by considering predictor-indicator bidirectional approximations. This type of approximations use a predictor function, that predicts the value of the function, and an indicator function, that indicates uncertainty about the predicted value. The advantage of this approach is that the predictor and indicator functions are not required to have implication relationships with the original

function $G$. However, predictor-indicator bidirectional approximations cannot be used when the bidirectional approximate circuit is vulnerable to errors [15].

Other implication-based approximation methods are proposed in [14], [17]–[20]. The approach in [18] considers the failure probabilities of the gates and uses a two-level representation. Finally, [19] approximates a circuit by removing circuit lines with low testability. However, this method does not allow to estimate the error probability produced by the approximation transformations. An improved probability estimation method is proposed in [20], but it does not take into account the possible faults that may occur in the approximate circuits. The probabilistic approach proposed in this work extends these techniques by considering a dynamic probability analysis and considering all faults that may occur in the original and the approximate circuits to estimate the total error probability.

### B. Evolutionary Circuit Design

Since the very beginning of the research in evolutionary computation, evolutionary algorithms have been applied for purposes of hardware optimization. Several monographs [21], [22] summarize the applications from the field of electronic circuits design, diagnostics, and testing. Later, evolutionary algorithms were applied to generate complete circuit structures (i.e., not only to optimize parameters of existing circuits) and dynamically adapt circuit structures [23]. For example, in the area of dependability, an evolutionary repair method was proposed for TMR implemented into FPGAs [24]. It employs an evolutionary algorithm to repair one damaged module of TMR by using the two healthy modules as sources of golden data for the fitness function. An analysis has shown a significant improvement of reliability for small benchmark circuits.

The evolutionary design of combinational circuits has been well established in the past. Majority of designs in this area is conducted by Cartesian genetic programming (CGP) or methods similar to CGP. CGP is a branch of genetic programming (GP) introduced by Miller and Thomson [25]. Unlike GP, which uses tree representation, an individual in CGP is represented by a directed acyclic graph of a fixed size. The candidate circuits can have multiple outputs and intermediate results can be reused (see details in Sect. IV). CGP can be used to design various types of circuits as surveyed in [26].

The trickiest component of the evolutionary circuit design is formulating the fitness function. It usually contains several objectives (functionality, area, delay etc.) where the functionality is typically understood as the quality of the candidate circuit measured as the number of correct output bits compared to a specified truth table (i.e. the Hamming distance). In order to obtain a fully working circuit, all combinations of input values have to be evaluated. For a circuit with $n_\mathrm{i}$ inputs and $n_\mathrm{o}$ outputs, $2^{n_\mathrm{i}}$ test vectors need to be fetched to the primary inputs and $n_\mathrm{o} \cdot 2^{n_\mathrm{i}}$ output bits have to be verified so as to compute the fitness value. The fitness calculation is computationally very intensive, since the number of test vectors grows exponentially with the number of primary inputs. Recently, it has been sped up by applying parallelism at various levels (data, thread, process) [27] or by introducing formal methods based on, for example, SAT solving [28].

When designing digital circuits with respect to multiple secondary objectives, e.g. area, latency, power consumption, or with the goal to approximate circuit behavior, one can make use of several approaches. The single-objective approach can be extended to deal with multiple objectives either by combining the objectives in a single fitness function just by summing the particular fitnesses weighted with a constant or, in a more sophisticated way, by introducing a multi-stage fitness function activating the particular objectives step by step. Thanks to the fixed size of the CGP genotype, resources can be constrained in order to find circuits with smaller area or power consumption [29]. Recently, a truly multi-objective approach to the design of (approximate) digital circuits has been proposed [30]. None of these methods, however, has been used to approximate TMR circuits.

## III. Probabilistic Generation of Approximate Logic Circuits

In our proposed approach, approximate logic circuits are obtained from the original circuit by iteratively performing some logic transformations. Note that these transformations are not required to preserve the original logic functionality, but rather to simplify the logic at the expense of deviating from the original behaviour and hence reduce the error coverage. Thus, the quality of an approximation transformation is characterized by two major parameters: the error probability increment and the area savings. Previous works mostly focus on the latter and use synthesis techniques to simplify the logic. However, the impact of approximations on the final error probability can hardly be estimated during the synthesis process and hence these methods offer limited scalability.

In our approach, the error probability and the synthesis transformations are linked through the stuck-at fault concept. The stuck-at fault model is commonly used to model permanent faults. Stuck-at fault simulation is also a common approach to estimate the error rate [11], [16]. For each fault, the error probability is estimated as the fraction of input vectors that test the fault. For a set $f_i$ of $N$ possible faults, the total error probability $EP$ can be computed as the average probability of testing every possible fault:

$$EP = \frac{\sum P(f_i)}{N}$$

This average can be weighted by the probability of each fault occurrence, if such information can be estimated. In the case of SETs, we can use derating factors to take into account timing and electrical masking.

On the other hand, the stuck-at fault concept is in the foundation of a class of powerful logic synthesis techniques [31], [32]. If a line stuck-at fault cannot be tested by any input vector, then the line is redundant and can be removed. Otherwise, if a line stuck-at fault is testable, the removal of the line creates a discrepancy with the original circuit. In exact synthesis, such discrepancy must be removed by adding some logic elsewhere [32], [33]. However, in approximate synthesis, the

discrepancy is allowed. A preliminary approach that exploits this technique has been proposed in [19].

### A. Line approximation

If a stuck-at fault in a line has low testability, it means that there are few input vectors that can test the fault. Then, a good approximate circuit can be built by assigning the line to a constant value. We refer to this transformation as line approximation. The error probability associated to this transformation is proportional to the probability of the input vectors which test the stuck-at fault. On the other hand, the area savings can be estimated as the logic that is removed by assigning the line, including the logic previously used to drive the line and the logic that can be simplified by propagating the constant value from the line.

The error produced by a line approximation is unidirectional if all the propagation paths from the line to the primary outputs have either an even or odd number of inversions. A line that meets this condition is said to have parity. The approximation of a line with no parity does not produce an under-approximation or an over-approximation. However, it is possible to approximate a circuit in lines with no parity by applying a simple transformation to the circuit. All lines in a circuit can be forced to have parity, except for possibly the primary inputs, by duplicating the gates and splitting the lines with no parity into two subsets with even and odd parities respectively [34]. This temporarily creates a larger circuit that allows the application of line approximations. Duplications that are not removed after approximation can be removed later on by resynthesizing the approximate logic circuit.

If a line $l$ with even parity is approximated by assigning it to a logic 0, then the incorrect subspace is made up of the input vectors that test the fault $l$ stuck-at 0. For these vectors, a $1 \rightarrow 0$ error is propagated without inversion to at least one output. Thus, the result is an under-approximation. If the line is assigned to a logic 1, then the incorrect subspace is made up of the input vectors that test the fault $l$ stuck-at 1. In this case a $0 \rightarrow 1$ error may be propagated without inversion and the result is an over-approximation. If the line $l$ has odd parity, the under-approximation and over-approximation are obtained with the opposite logic assignments. In conclusion, the two stuck-at faults of each line in a unidirectional circuit can be associated to either the under-approximation or the over-approximation, respectively.

Fig. 3 shows a logic circuit example and its K-map. Two approximations are shown on the right of the figure. The first one is obtained by making $d = 1$ and the incorrect subspace is highlighted in the resulting K-map. The result is an over-approximation because all errors are $0 \rightarrow 1$ errors. The second one is obtained by making $g_1 = 0$. This is an under-approximation because all errors are $1 \rightarrow 0$ errors.

Approximate circuits can be built in this way by selecting a subset of lines for approximation. In [19], all lines whose fault probability is below a selected threshold are approximated. In this work we follow an iterative approach. In each iteration, the least testable fault is selected for approximation and the effect of the line approximation on the error probability $EP$ and the area are estimated. Iterations continue until the error probability or the area estimation reach the required target. The approximation space can be traversed with fine resolution, because each single line approximation produces a small impact on both the error probability and the area.

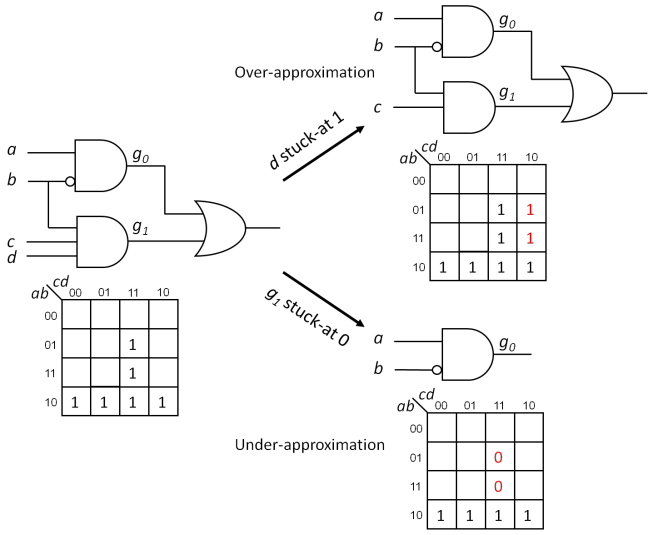



Fig. 3. Circuit approximation example

### B. Dynamic Probability Analysis

It must be noted that after performing an approximation by making a line of the circuit constant, the testability of the remaining faults change. This can be illustrated with a very simple example.

Consider a 4-input AND gate, as shown in Fig. 4, and assume all of the 16 input vectors are equally likely. For a stuck-at 1 fault at any of the inputs, there is only one input vector that can test the fault, so that the detection probability is 1/16. We can then approximate the circuit by forcing one of the lines to a constant 1. The resulting approximate circuit is a 3-input AND gate. If we want to approximate additional inputs, we must take into account that the detection probability of the stuck-at 1 faults at the remaining inputs is no longer 1/16 but 1/8. Subsequent approximations at the inputs will further increase the probability to 1/4, 1/2 and 1 (when all the inputs are removed). Thus, the initial error probability estimation is less and less accurate as more approximations are taken. This requires a dynamic probability update every time an approximation is taken.

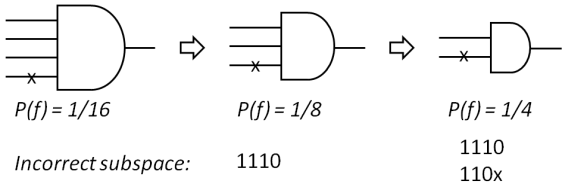



Fig. 4. Dynamic probability with successive approximations.

The problem of estimating the probability of testing a fault is generally related to testability analysis. The goal

of testability analysis is to measure the difficulty of testing a fault. This measure is typically used to identify hard-to-test faults and to predict the quality of random test pattern generation. Many algorithms for testability analysis have been proposed [35]–[38] which mainly use probabilistic approaches. As the estimation of probabilities has to be updated for every circuit approximation, we need a fast and incremental algorithm. The approach used in this work was originally proposed in [20]. We summarize it here for the sake of completeness.

The notions of signal probability were established by Parker and McCluskey in [39] and are well-known in the literature. The probability that a stuck-at fault is detected by a random input vector, also called detectability, can be formulated in terms of signal probabilities [40]. The detection of a fault requires an input vector that is able to set a control value at the fault site (controllability) and to sensitize at least one propagation path (observability). These conditions can be expressed as Mandatory Assignments (MAs), so that the probability of testing a fault $f$ can be computed as the joint probability of these MAs [41].

For instance, let us consider again the example in Fig. 3 and the fault $d$ stuck-at 1. For this fault, the controllability assignment is $d = 0$ and the observability assignments are $b = 1$, $c = 1$ and $g_0 = 0$. Thus, the probability of testing this fault can be expressed as

$$P(f) = P(\mathrm{SMA}_f) = P(b = 1, c = 1, d = 0, g_0 = 0)$$

where $\mathrm{SMA}_f$ is the Set of Mandatory Assignments for the fault $f$. From here on we will use indistinctly $P(f)$ or $P(\mathrm{SMA}_f)$ to denote the probability of testing a fault $f$.

Signal probabilities in a combinational network can be easily computed by traversing the circuit from inputs to outputs [39]. At each node, the signal probability of the output is obtained as a function of the probability of the input signals according to the node type. Fault detectability can be computed as the product of the probabilities of all MAs [42]. However, this approach is only correct if all MAs are independent. To improve the accuracy of probability estimations, we use implication reasoning. Implications are actually a consequence of the existence of signal dependencies, so that signal dependencies can be removed by implying the MAs backward and forward. Then, the fault detection probability is computed as the joing probability of the final set of backward implications which cannot be further justified. If the implication of the SMA leads to an inconsistency, then we can conclude that the fault is redundant and its probability is 0. The experimental results shown in [20] demonstrate that this approach produces good estimations in comparison with stuck-at fault simulation.

In the example of Fig. 3, let us consider now the fault $g_1$ stuck-at 0. In this case we have $\mathrm{SMA}_f = \{g_0 = 0, g_1 = 1\}$. If all inputs are equally likely, then $P(g_0 = 0) = 0.75$, $P(g_1 = 1) = 0.125$ and $P(f) = P(\mathrm{SMA}_f) = 0.75 \cdot 0.125 = 0.09375$. This result is not accurate, but we can use implications to obtain the correct probability. In particular, $g_1 = 1$ implies $b = c = d = 1$ and $b = 1$ justifies $g_0 = 0$. Therefore, $\mathrm{SMA}_f = \{b = 1, c = 1, d = 1\}$. The product of the probabilities of these MAs gives the correct result $P(f) = 0.125$.

### C. Probability-based approximation

To generate approximate logic circuits, we depart from a TMR circuit using three exact copies of the original circuit. In this circuit, when no approximation has been taken yet, any error is masked by the voter and $EP = 0$.

Consider the circuit in Fig. 1 which consists of the original circuit $G$, an under-approximation $F$ and an over-approximation $H$. $F$ and $H$ have been obtained from $G$ by approximating some lines. Let $A_F$ and $A_H$ be the set of faults that have been approximated to obtain $F$ and $H$, respectively. The incorrect subspace is the set of input vectors that test a fault in $A_F$ or $A_H$. Note that the test vectors for the faults in $A_F$ and $A_H$ do not overlap. Because of the approximations, some input vectors can produce a $0 \rightarrow 1$ error in $F$ and some others can produce a $1 \rightarrow 0$ error in $H$, but it is guaranteed by construction that at least one of the two approximate circuits produces the correct value. Therefore, no error is observed in the absence of faults even though one of the approximations may produce a wrong value for some input vectors.

When a fault occurs in one of the three circuits, it may produce an error. In the correct subspace, this error is masked because the other two circuits are correct. However, in the incorrect subspace, one of the two approximate circuits, $F$ or $H$, is also producing an error. Therefore, two of the three circuits are wrong and the error is unmasked. More precisely, errors may happen in the following three cases:

1) A fault $f_G$ in the original circuit $G$ may produce an error only if it propagates to the output for an input vector that tests a fault in $A_F$ or $A_H$. The error probability in this case is $P(f_G \cap (A_F \cup A_H))$. Because $A_F$ and $A_H$ are disjoint, this probability can be computed as the sum of two terms, $P(f_G \cap A_F)$ + $P(f_G \cap A_H)$.
2) A fault $f_F$ in $F$ that produces an error $0 \rightarrow 1$ in $F$ for an input vector that tests a fault in $A_H$. The error probability in this case is $P(f_F \cap A_H)$.
3) A fault $f_H$ in $H$ that produces an error $1 \rightarrow 0$ in $H$ for an input vector that tests a fault in $A_F$. The error probability in this case is $P(f_H \cap A_F)$.

Note that faults that occur in an approximate circuit, $F$ or $H$, may contribute to an unmasked error only in one direction. It is guaranteed by construction that faults that occur in the opposite direction either correct the error created by the approximation or cannot propagate in the incorrect subspace. On the other hand, the number of faults in $F$ and $H$ becomes smaller as more approximations are performed. As a consequence, the contribution of the last two cases is typically smaller than the first one. Notwithstanding, we consider all three cases in our algorithm.

To compute the probabilities in each case, we keep the SMA of each approximated line. Then, the probability of the incorrect subspaces given by $A_F$ and $A_H$ can be computed as the probability of the union of these SMAs. When a new approximation is performed, the new SMA is added to the set.

Fig. 5 shows the pseudo-code of the approximation algorithm. In the initialization step, we create $F$ and $H$ which are exact copies of the original circuit, so the total error probability $EP$ is 0 and the estimated area $EA$ is three times that of

**Inputs**: Original circuit (G), Estimated Area (EA),
Error Probability Target (EP_T), Area Target (EA_T)
**Outputs**: Under-approximate circuit (F), Over-approximate circuit (H),
Estimated Error Probability (EP), Estimated Final Area (EA)
// Initialization
$F = G$, $AF = \phi$, $EA_F = EA$
$H = G$, $AH = \phi$, $EA_G = EA$
$EP = 0$
$EA = EA_F + EA_G + EA_H$
// Compute initial fault probabilities
**foreach** fault $f_i$ in $G$
Imply $SMA(f_i)$
Compute $P(f_i)$
Copy $SMA(f_i)$ and $P(f_i)$ to equivalent faults in $F$ and $H$
// Approximation loop
**while** ($EP < EP\_T$ ***or*** $EA > EA\_T$){
select fault $f_A$ with min probability;
**if** $f_A$ is an under-approximation fault **then**
$A_F = A_F \cup f_A$
**foreach** fault $f_G$ in G
$P(f_G) = P(f_G \cap A_F)$
**foreach** fault $f_H$ in $H$ that propagates as $1 \rightarrow 0$
$P(f_H) = P(f_H \cap A_F)$
Update $EP$
Approximate $f_A$ in $F$
**foreach** fault $f_F$ in $F$
Update $SMA(f_F)$
**if** $f_F$ is redundant, **then** approximate $f_F$ in $F$
Update $EA$
**else** // $f_A$ is an over-approximation fault
$A_H = A_H \cup f_A$
**foreach** fault $f_G$ in G
$P(f_G) = P(f_G \cap A_H)$
**foreach** fault $f_F$ in $F$ that propagates as $0 \rightarrow 1$
$P(f_F) = P(f_F \cap A_H)$
Update $EP$
Approximate $f_A$ in $H$
**foreach** fault $f_H$ in $H$
Update $SMA(f_H)$
**if** $f_H$ is redundant, **then** approximate $f_H$ in $H$
Update $EA$
}

Fig. 5. Probabilistic approximation algorithm

the original circuit. We also compute the initial SMA of each fault and the fault probabilities, which are equivalent in the three circuits. Then, we enter the approximation loop. In each iteration, we select for approximation the fault $f_A$ with the minimum probability. This fault can result either in an under-approximation or in an over-approximation. Depending on the type of fault, we add it to $A_F$ or $A_H$, respectively. Then we compute the new fault probabilities for all possible faults and update the total probability estimation. Note that if the fault results in an under-approximation, the second case does not apply and only one of the terms in the first case needs to be computed because the other does not vary. Similarly, if the fault results in an under-approximation, the third case does not apply and only the other term in the first case needs to be computed. Finally, the approximation is performed and we update the SMA of all faults in the approximate circuit. Along this process we also eliminate possible redundancies that the approximation may have created elsewhere. The process is repeated until the EP target and the EA target are met.

## IV. Evolutionary design of approximate logic circuits

The proposed method is based on CGP, in which a circuit is represented as a fixed-sized cartesian grid of $n_\mathrm{r} \times n_\mathrm{c}$ nodes interconnected by a feed-forward network (see Figure 6). Node inputs can be connected either to one of $n_\mathrm{i}$ primary inputs or to an output of a node in preceding $L$ columns. Each node has a fixed number of inputs $n_\mathrm{a}$ (usually $n_\mathrm{a} = 2$) and can perform one of the logic functions from a predefined set $\Gamma$. Each of $n_\mathrm{o}$ primary circuit outputs can be connected either to a primary input or to a node's output. The area and delay of the circuit can be constrained by changing the grid size and the $L$-back parameter.

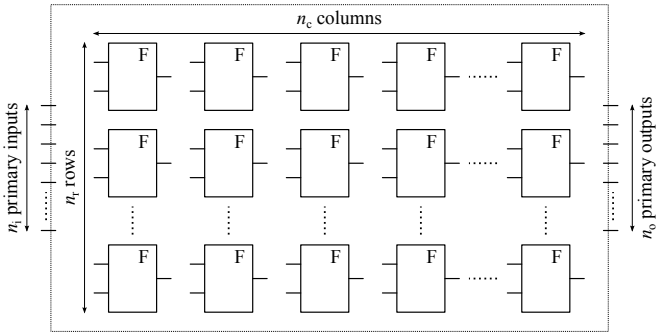



Fig. 6. Cartesian genetic programming schema.

While the search is conducted at the level of genotypes (arrays of integers representing the circuit), the fitness function evaluates phenotypes (circuits established according to the genotypes). The actual encoding is as follows: The primary inputs and the outputs of nodes are labeled $0 \ldots n_c \cdot n_r + n_i - 1$ and considered as addresses which connections can be fed to. In the genotype, each two-input node is then encoded using three integers (an address for the first input; an address for the second input; a node function). Finally, for each primary output, the genotype contains one integer specifying the connection address. The genotype size is $(n_a+1)n_r n_c + n_o$ genes (integers). While the genotype is of fixed length, the size of the phenotype depends on the number of inactive nodes, i.e. nodes whose output is not used by any other node or primary output. Since the inactive nodes have no influence on the phenotype, there are individuals with different genotypes but the same phenotypes.

An example of a CGP individual with its chromosome can be seen in Figure 7. It has three inputs, one output and three active nodes.

CGP uses a simple mutation based $(1 + \lambda)$ evolutionary strategy as a search mechanism. The population size $1 + \lambda$ is mostly very small, typically, $\lambda = 4$. The maximum number of generations created in a single run is $N_\mathrm{g}$. The initial population

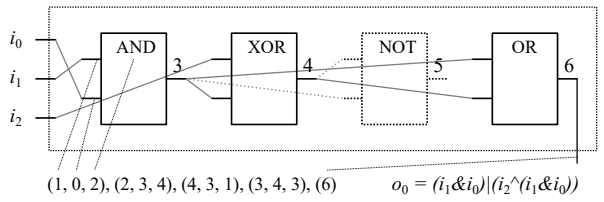



Fig. 7. CGP individual example.

is constructed either randomly (in the case of evolutionary design) or by mapping of a known solution to the CGP chromosome (in the case of evolutionary optimization). In each generation, the best individual is passed to the next generation unmodified along with its $\lambda$ offspring individuals created by means of a point mutation operator. In case more individuals with the best fitness exist, a randomly selected one is chosen. The mutation rate $m$ is usually set to modify up to 5 % randomly selected genes. The role of mutation is significant in CGP (see detailed analysis in [43], [44]).

Based on our previous experiences, we decided to use a multi-stage single-objective approach with constrained resources to obtain desired approximations. The fitness function $f_{\text{under}}$ used to find under-approximations is defined as follows:

$$f_{\text{under}} := \begin{cases} f_{\text{hamm}}^{\max} + (f_{\text{area}}^{\max} - f_{\text{area}}) & \text{if } f_{\text{hamm}} = 0, \\ f_{\text{hamm}}^{\max} - f_{\text{hamm}} & \text{if } f_{\text{off}} = 0, \\ f_{\text{off}}^{\max} - f_{\text{off}} & \text{otherwise,} \end{cases} \quad (1)$$

where $f_{\text{hamm}}$ is the total Hamming distance between the outputs generated by the candidate solution and the original circuit for all possible input combinations, $f_{\text{hamm}}^{\max} = n_{\text{o}} 2^{n_{\text{i}}}$, $f_{\text{area}}$ is the area of the circuit, $f_{\text{area}}^{\max}$ is the maximum area according to chosen number of rows $n_{\text{r}}$ and columns $n_{\text{c}}$, $f_{\text{off}}$ is the number of $0 \rightarrow 1$ errors for all possible input combinations and $f_{\text{off}}^{\max}$ is the number of zeros in the truth table of the original circuit. All individuals with fitness $f_{\text{under}} \geq f_{\text{off}}^{\max}$ represent a valid under-approximation.

Similarly, the fitness function $f_{\text{over}}$ used to find over-approximations is defined as follows:

$$f_{\text{over}} := \begin{cases} f_{\text{hamm}}^{\max} + (f_{\text{area}}^{\max} - f_{\text{area}}) & \text{if } f_{\text{hamm}} = 0, \\ f_{\text{hamm}}^{\max} - f_{\text{hamm}} & \text{if } f_{\text{on}} = 0, \\ f_{\text{on}}^{\max} - f_{\text{on}} & \text{otherwise,} \end{cases} \quad (2)$$

where $f_{\text{on}}$ is the number of $1 \rightarrow 0$ errors for all possible input combinations and $f_{\text{on}}^{\max}$ is the number of ones in the truth table of the original circuit. All individuals with fitness $f_{\text{over}} \geq f_{\text{on}}^{\max}$ represent a valid over-approximation. One can observe that since all possible input vectors have to be generated, the approach is not scalable. In order to speed up the design, the parallelism at various levels (data, thread, process) can be introduced [27]. In practice, it is applicable for circuits containing less than approximately 20 inputs and 200 gates. More complex circuits can be optimized by introducing formal methods, e.g. SAT solvers or Binary Decision Diagrams (BDD), however, an initial fully working solution is needed in this case [45].

## V. Experimental Results

### A. Experimental setup

The experiments were conducted with two groups of benchmarks extracted from LGSynth93 set. The first group was intended to compare the results of both probabilistic and evolutionary approaches. Therefore, the size of the circuits in this group was limited to the capabilities of the evolutionary approach. With the second group of benchmarks the goal was to show the efficiency of the probabilistic approach for other circuits and to compare it with other approaches. Benchmarks b12, rd73 and t481 were selected for the first group, and apex3, apex4, m3, misex3, table3 and table5 for the second. The original version of each benchmark was obtained by synthesizing the circuit with Synopsys using the Nangate15nm synthesis library [46]. Table I shows the size of each benchmark as provided by the synthesis tool both in the number of cells and the area, as well as the number of primary inputs (PIs) and outputs (POs).

TABLE I
Synthesis results for selected benchmarks

| Benchmark | #PIs | #POs | #cells | area |
|---|---|---|---|---|
| apex3 | 54 | 50 | 799 | 192.77 |
| apex4 | 9 | 18 | 1191 | 279.77 |
| b12 | 15 | 9 | 58 | 12.93 |
| m3 | 8 | 16 | 205 | 46.55 |
| misex3 | 14 | 14 | 1285 | 290.14 |
| rd73 | 7 | 3 | 20 | 5.90 |
| t481 | 16 | 1 | 32 | 6.98 |
| table3 | 14 | 14 | 478 | 116.59 |
| table5 | 17 | 15 | 420 | 103.61 |

The generation of approximate logic circuits for the experiments was done in the following way. For the probabilistic approach, a set of arbitrary error targets was set for each circuit, covering a significant range of cases between conventional TMR (full protection) and trivial approximation (no redundant logic). Each error target generated a pair of approximate circuits (over-approximation and under-approximation), which were resynthesized in order to remove logic constants. Each pair of approximate circuits was combined with the original circuit to build an error masking schema.

With respect to the evolutionary approach, a large set of approximate circuits was generated for each benchmark. Table II summarizes the CGP parameters whose values were set up as recommended in the literature [26].

TABLE II
CGP parameters

| Parameter | b12 | rd73 | t481 |
|---|---|---|---|
| $n_{\text{i}}$ | 15 | 7 | 16 |
| $n_{\text{o}}$ | 9 | 3 | 1 |
| $n_{\text{c}}$ | 6 | 5 | 5 |
| $n_{\text{r}}$ | | $1 \ldots 10$ | |
| $L$ | 6 | 5 | 5 |
| $\Gamma$ | | all 2-input gates | |
| $\lambda$ | | 4 | |
| $m$ | | 5 % | |
| $N_{\text{g}}$ | 2000000 | 2000000 | 1000000 |

For each configuration of the CGP grid (as $n_{\text{r}}$ varies from 1 to 10), a total of 100 over-approximations and 100 under-

approximations were generated. In total, 1000 approximate circuits of each type were generated for each benchmark. According to the fundamentals of the proposed technique, any over-approximation can be combined with any under-approximation to conform a valid error masking scheme. Therefore, there are $10^6$ possible solutions to test for each benchmark. Testing the whole set of solutions would take too much time, and therefore it was studied if there was any representative data that allowed to select the best solutions in terms of error masking capabilities. This is discussed in section V-B. Figures 8 and 9 show, in the form of box plots, a statistical analysis of multiple CGP runs for selected circuits evolved as the under-approximations and over-approximations. The box plots give the Hamming distances obtained for increased amount of resources available for the implementation. A clear trade-off between the Hamming distance (quality) and the area can be observed.

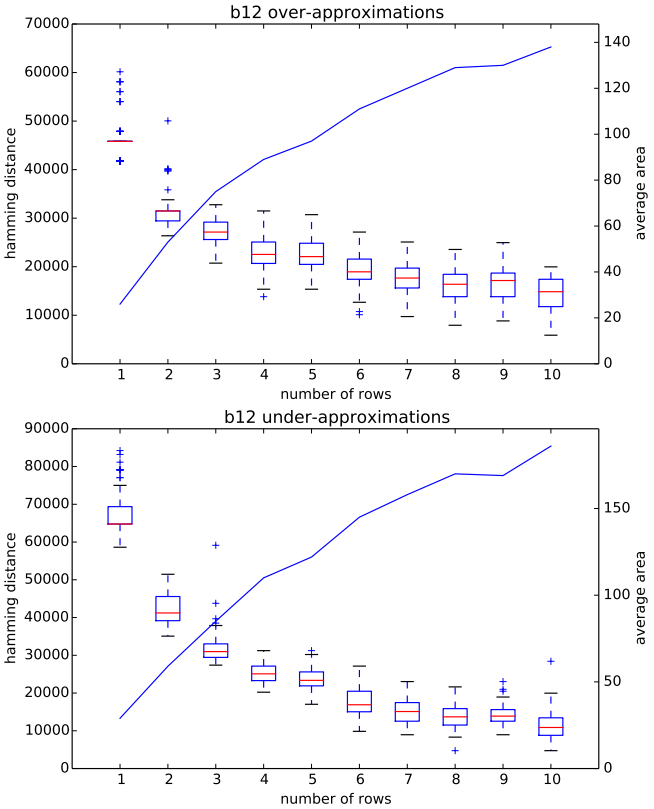



Fig. 8. Statistical results for b12 approximations evolved by CGP.

Once approximate circuits were generated, masking schemas were built for testing. Voters were placed at the output of circuits, and the list of stuck-at faults was generated for each circuit. This list included all faults on every input of each gate in the circuit, plus the faults on the outputs of the circuit before the voter. This allowed to introduce simple voters, as there is full control of fault injection points.

For each error masking schema under test, a fault simulation with random input vectors was performed by means of the parallel simulator HOPE [47]. A total of 50000 randomly generated input vectors were applied for each design under

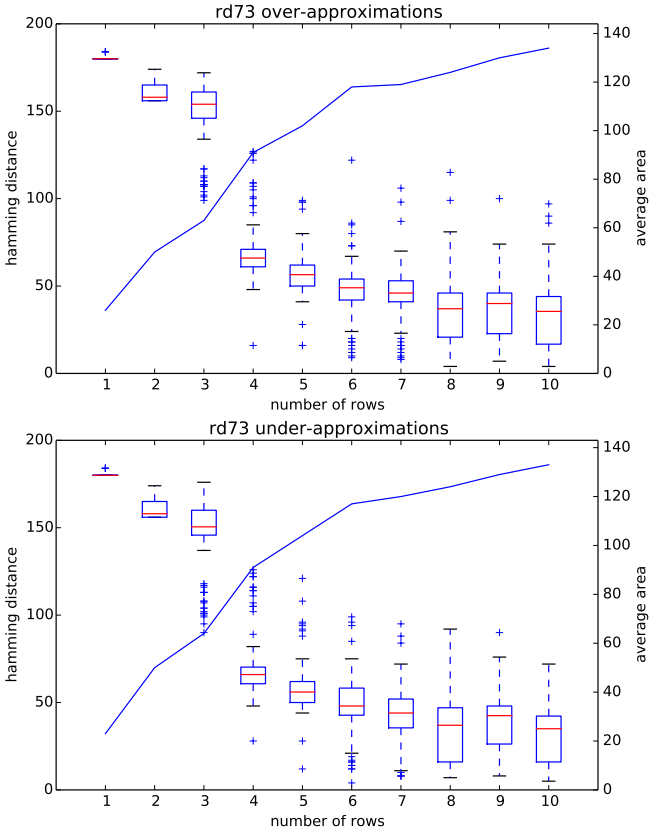



Fig. 9. Statistical results for rd73 approximations evolved by CGP.

test, and all faults in the list previously generated were tested for each input vector. The total error probability was computed as the average number of faults detected per input vector, divided by the size of the fault list. For simplicity, all faults were considered equally likely.

At the time of estimating the probability of circuit failure, it must be taken into account that the number of faults hitting a circuit is correlated with the circuit area. Thus, as the area increases by using larger approximations, the fault probability increases as well. However, all faults in the approximate circuits are either masked or detectable, so that the probability of circuit failure always decreases as the quality of the approximation increases.

### *B. Selection of candidate Approximate Logic Circuits from evolutionary approach*

As stated before, 1000 approximate circuits of each type were generated with the evolutionary approach for the experiments, which made a total of $10^6$ combinations to test. For the first benchmark (b12) an exhaustive analysis was performed. The whole process is very time consuming, therefore data collected for b12 were studied in order to properly select the most promising candidates for the rest of the benchmarks. The objective was to find the combination with the highest error masking rate. The more functionally similar are the approximate circuits with respect to the original circuit, the more protection against faults is achieved. Therefore, approx-

imate circuits with low Hamming distance compared with the original circuit are good candidates, in principle. To validate this hypothesis, the correlation between the sum of Hamming distances of both approximate circuits and the experimental error probability was computed. The results are shown in Table III, grouped according to the size of the configuration matrix for both under- and over-approximate circuits. The results show that there is a significant correlation between both metrics. The average correlation index is 0.831. Therefore, for the rest of the benchmarks only the circuits with a Hamming distance below the average of each group were selected for experiments.

### *C. Comparison between probabilistic and evolutionary approaches*

First, the results of experiments aimed to compare both techniques are shown here. Fig. 10 graphically represents the tradeoff between error probability and area overhead for several solutions found by using either the probabilistic or the evolutionary approach. This probability is related to the number of faults in the original circuit in order to take into account the area increase. For the latter technique, only the cases with the best error masking rate for each possible combination in the sizes of under-approximate and over- approximate circuits are represented. The same applies for Fig. 11 and 12 with respect to rd73 and t481 benchmarks, respectively.

Analyzing the results in Fig. 10, it can be seen that the evolutionary approach achieves in general slightly better results than the probabilistic one for b12 benchmark. This is reasonable, because evolutionary approach can explore a much larger range of solutions, although at a much larger computational cost. However, the probabilistic approach can still obtain good solutions, close to the evolutionary approach.

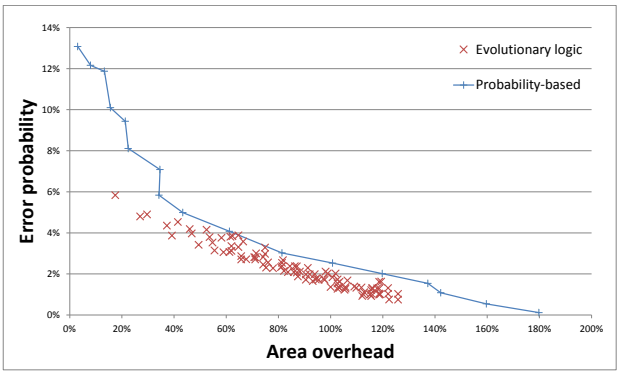



Fig. 10. b12 Simulation Results

Results for rd73 benchmark are shown on Fig. 11. This is an example of a circuit with a high degree of binateness, which means that small expansions on either the on-set or the off-set have a high cost in terms of resources. This leads to sub-optimal solutions with overheads greater than 200 % in both approaches, which are uninteresting. On the other hand, under the 200 % overhead limit the same tendency is observed. The evolutionary approach produces slightly better solutions than the probabilistic approach, but with much more computational effort.

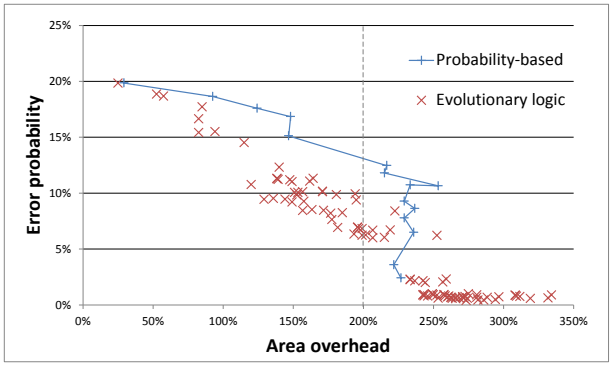



Fig. 11. rd73 Simulation Results

With respect to t481 benchmark (Fig. 12), it can be observed that the probabilistic approach presents more scalability than the evolutionary one. This is due to the fact that t481 is a circuit with just one output with high onset probability. Under such conditions, the evolutionary approach tends to generate onset circuits which behave like logic constants due to the limited size of the configurable logic array, thus limiting both area overhead and error masking rate. On the other hand, the probabilistic approach is based on gradually degrading the logic function of the circuit, which allows to reach more robust solutions in the region close to conventional TMR.

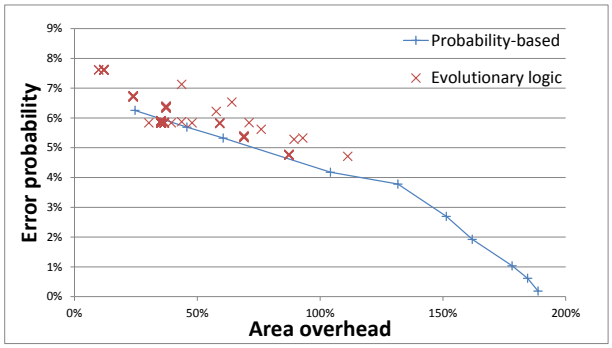



Fig. 12. t481 Simulation Results

### *D. Comparison with other approaches*

A second group of experiments was performed with the aim of showing the applicability and scalability of the probabilistic approach for larger circuits. In addition, the results from this group were compared with the sub-circuit resynthesizing method recently proposed in [11]. The results of the experiments are shown in Fig. 13. The graphics show for each benchmark the tradeoff between the $EP$ improvement, i.e., the improvement in the $EP$ with respect to the original

TABLE III
ERROR MASKING RATE VS. HAMMING DISTANCE CORRELATION INDEXES

| Over/Under | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.700 | 0.686 | 0.739 | 0.754 | 0.708 | 0.707 | 0.674 | 0.646 | 0.688 | 0.667 |
| 2 | 0.792 | 0.772 | 0.803 | 0.843 | 0.818 | 0.840 | 0.838 | 0.820 | 0.845 | 0.839 |
| 3 | 0.717 | 0.626 | 0.653 | 0.732 | 0.704 | 0.766 | 0.770 | 0.763 | 0.786 | 0.797 |
| 4 | 0.802 | 0.734 | 0.820 | 0.875 | 0.833 | 0.867 | 0.890 | 0.879 | 0.887 | 0.875 |
| 5 | 0.806 | 0.737 | 0.799 | 0.847 | 0.815 | 0.849 | 0.867 | 0.857 | 0.868 | 0.861 |
| 6 | 0.836 | 0.818 | 0.871 | 0.903 | 0.885 | 0.900 | 0.916 | 0.910 | 0.914 | 0.913 |
| 7 | 0.823 | 0.814 | 0.861 | 0.897 | 0.882 | 0.897 | 0.913 | 0.905 | 0.911 | 0.912 |
| 8 | 0.811 | 0.805 | 0.853 | 0.893 | 0.880 | 0.896 | 0.912 | 0.902 | 0.911 | 0.914 |
| 9 | 0.795 | 0.751 | 0.795 | 0.854 | 0.834 | 0.867 | 0.887 | 0.875 | 0.888 | 0.893 |
| 10 | 0.815 | 0.840 | 0.865 | 0.901 | 0.900 | 0.913 | 0.929 | 0.923 | 0.930 | 0.938 |

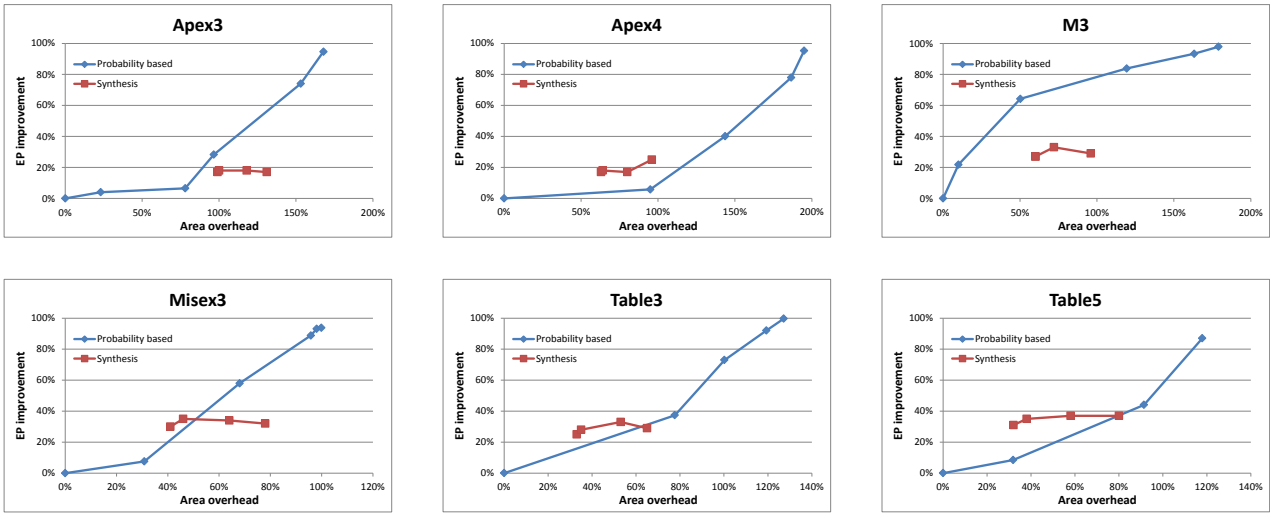



Fig. 13. Simulation results and comparison with synthesis-based approach [11]

unprotected circuit, and the area overhead for different targets. For the probabilistic approach, the additional area due to the approximate circuits and single voters is taken into account. The graphics also include data from [11] for the sake of comparison. It must be noted that these data were obtained with a different synthesis tool and a different technology. Notwithstanding, with the necessary precautions, the results can be used for a general comparison.

The results show that the synthesis approach tends to produce slightly more efficient solutions with respect to the probabilistic approach, although the probabilistic approach can produce better results in some cases. On the other hand, the synthesis approach evinces a very limited scalability, while the probabilistic approach is able to reach any level of error protection, as high as desired. As a matter of fact, the synthesis approach cannot significantly improve the $EP$ by increasing the area overhead.

Finally, it must be noted that a single particle strike can generate a multiple fault in adjacent cells due to charge sharing. This effect is critical in the synthesis approach, as a multiple fault can invalidate the logical masking. However, our approach provides protection against this effect by construction, because the three circuits are built separately. Thus, a multiple fault caused by charge sharing can only affect one of the circuits and the multiple error can be masked at the voter.

## VI. CONCLUSIONS

In this work we proposed and compared two different approaches to generate approximate logic circuits for error mitigation using a TMR schema. The probablilistic approach uses a greedy algorithm based on line approximations and dynamic error probability estimations. The evolutionary approach is based on CGP and can generate radically different solutions. The experimental results show that the evolutionary approach is generally able to find slightly better results, but at the expense of a higher computational effort. On the other hand, the probabilistic approach can handle large circuits in an efficient manner. Notwithstanding, the current progress in evolutionary computing techniques suggests that they will be able to process larger circuits in the near future [45].

The two proposed methods are widely scalable and can provide solutions for any required trade-off between reliability

and area overhead. This is a major advantage to cover a variety of application scenarios and technologies. In comparison, recently proposed synthesis-based methods can occasionally produce slightly better results but they cannot explore the design space in depth.

## REFERENCES


[1] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *Device and Materials Reliability, IEEE Transactions on*, vol. 5, no. 3, pp. 305–316, 2005.

[2] G. Neuberger, G. Wirth, and S. O. service), *Protecting Chips Against Hold Time Violations Due to Variability*. Dordrecht :: Springer Netherlands :, 2014. [Online]. Available: http://dx.doi.org/10.1007/978-94-007-2427-3

[3] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *Micro, IEEE*, vol. 25, no. 6, pp. 10–16, 2005.

[4] V. Ferlet-Cavrois, L. W. Massengill, and P. Gouker, "Single event transients in digital cmosâĂŤa review," *Nuclear Science, IEEE Transactions on*, vol. 60, no. 3, pp. 1767–1790, 2013.

[5] M. M. Shulaker, G. Hills, N. Patil, H. Wei, H.-Y. Chen, H.-S. P. Wong, and S. Mitra, "Carbon nanotube computer," *Nature*, vol. 501, no. 7468, pp. 526–530, 2013.

[6] I. Polian and J. P. Hayes, "Selective hardening: Toward cost-effective error tolerance," *IEEE Design & Test of Computers*, no. 3, pp. 54–63, 2010.

[7] B. W. Johnson, *Design & analysis of fault tolerant digital systems*. Addison-Wesley Longman Publishing Co., Inc., 1988.

[8] K. Mohanram, N. Touba *et al.*, "Partial error masking to reduce soft error failure rate in logic circuits," in *Defect and Fault Tolerance in VLSI Systems, 2003. Proceedings. 18th IEEE International Symposium on*. IEEE, 2003, pp. 433–440.

[9] A. H. El-Maleh and F. C. Oughali, "A generalized modular redundancy scheme for enhancing fault tolerance of combinational circuits," *Microelectronics Reliability*, vol. 54, no. 1, pp. 316–326, 2014.

[10] K. Nepal, N. Alves, J. Dworak, and R. I. Bahar, "Using implications for online error detection," in *Test Conference, 2008. ITC 2008. IEEE International*. IEEE, 2008, pp. 1–10.

[11] A. El-Maleh and K. Daud, "Simulation-based method for synthesizing soft error tolerant combinational circuits," *Reliability, IEEE Transactions on*, vol. PP, no. 99, pp. 1–14, 2015.

[12] S. Krishnaswamy, S. M. Plaza, I. L. Markov, and J. P. Hayes, "Enhancing design robustness with reliability-aware resynthesis and logic simulation," in *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*. IEEE, 2007, pp. 149–154.

[13] S. Almukhaizim and Y. Makris, "Soft error mitigation through selective addition of functionally redundant wires," *Reliability, IEEE Transactions on*, vol. 57, no. 1, pp. 23–31, 2008.

[14] B. D. Sierawski, B. L. Bhuva, and L. W. Massengill, "Reducing soft error rate in logic circuits through approximate logic functions," *Nuclear Science, IEEE Transactions on*, vol. 53, no. 6, pp. 3417–3421, 2006.

[15] M. Choudhury and K. Mohanram, "Low cost concurrent error masking using approximate logic circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 32, no. 8, pp. 1163–1176, Aug 2013.

[16] M. R. Choudhury and K. Mohanram, "Approximate logic circuits for low overhead, non-intrusive concurrent error detection," in *Design, Automation and Test in Europe, 2008. DATE'08*. IEEE, 2008, pp. 903–908.

[17] I. Gomes, M. Martins, F. Lima Kastensmidt, A. Reis, R. Ribas, and S. Novales, "Methodology for achieving best trade-off of area and fault masking coverage in atmr," in *Test Workshop - LATW, 2014 15th Latin American*, March 2014, pp. 1–6.

[18] H. Xie, L. Chen, R. Liu, A. Evans, D. Alexandrescu, S.-J. Wen, and R. Wong, "New approaches for synthesis of redundant combinatorial logic for selective fault tolerance," in *On-Line Testing Symposium (IOLTS), 2014 IEEE 20th International*, July 2014, pp. 62–68.

[19] A. Sanchez-Clemente, L. Entrena, M. Garcia-Valderas, and C. Lopez-Ongil, "Logic masking for set mitigation using approximate logic circuits," in *On-Line Testing Symposium (IOLTS), 2012 IEEE 18th International*, June 2012, pp. 176–181.

[20] A. Sanchez-Clemente, L. Entrena, and M. Garcia-Valderas, "Error masking with approximate logic circuits using dynamic probability estimations," in *On-Line Testing Symposium (IOLTS), 2014 IEEE 20th International*, July 2014, pp. 134–139.

[21] R. Drechsler, *Evolutionary Algorithms for VLSI CAD*. Boston: Kluwer Academic Publishers, 1998.

[22] E. Larsson, *Introduction to Advanced System-on-Chip Test Design and Optimization*. Springer, 2005.

[23] M. A. Trefzer and A. M. Tyrrell, *Evolvable Hardware: From Practice to Application*. Springer, 2015.

[24] M. Garvie and A. Thompson, "Scrubbing away transients and jiggling around the permanent: long survival of fpga systems through evolutionary self-repair," in *Proc of the 10th IEEE International On-Line Testing Symposium IOLTS 2004*, 2004, pp. 155–160.

[25] J. Miller and P. Thomson, "Cartesian genetic programming," in *Genetic Programming*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2000, vol. 1802, pp. 121–132. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-46239-2_9

[26] J. F. Miller, Ed., *Cartesian Genetic Programming*, ser. Natural Computing Series. Springer Verlag, 2011. [Online]. Available: http://www.fit.vutbr.cz/research/view_pub.php?id=7299

[27] R. Hrbacek and L. Sekanina, "Towards highly optimized cartesian genetic programming: From sequential via simd and thread to massive parallel implementation," in *GECCO '14 Proceedings of the 2014 conference on Genetic and evolutionary computation*. Association for Computing Machinery, 2014, pp. 1015–1022.

[28] Z. Vasicek and L. Sekanina, "Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware," *Genetic Programming and Evolvable Machines*, vol. 12, no. 3, pp. 305–327, 2011.

[29] ——, "Evolutionary approach to approximate digital circuits design," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, 2015. [Online]. Available: http://www.fit.vutbr.cz/research/view_pub.php?id=10406

[30] R. Hrbacek, "Parallel multi-objective evolutionary design of approximate circuits," in *GECCO '15 Proceedings of the 2014 conference on Genetic and evolutionary computation*. Association for Computing Machinery, 2015, pp. 687–694.

[31] M. Iyer, M. Abramovici *et al.*, "Fire: a fault-independent combinational redundancy identification algorithm," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 4, no. 2, pp. 295–301, 1996.

[32] L. Entrena, K.-T. Cheng *et al.*, "Combinational and sequential logic optimization by redundancy addition and removal," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 14, no. 7, pp. 909–916, 1995.

[33] S.-C. Chang, M. Marek-Sadowska, and K.-T. Cheng, "Perturb and simplify: multilevel boolean network optimizer," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 15, no. 12, pp. 1494–1504, 1996.

[34] H. Kim and J. Hayes, "Realization-independent atpg for designs with unimplemented blocks," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 20, no. 2, pp. 290–306, Feb 2001.

[35] F. Brglez, "On testability of combinational networks," in *IEEE International Symposium on Circuits and Systems*, 1984.

[36] S. C. Seth, L. Pan, and V. D. Agrawal, "PREDICT-probabilistic estimation of digital circuit testability," in *Proceeding of International Symposium on Fault-Tolerant Computing*, Jun. 1985, pp. 220–225.

[37] S. Chakravarty and I. Hunt, H.B., "On computing signal probability and detection probability of stuck-at faults," *Computers, IEEE Transactions on*, vol. 39, no. 11, pp. 1369–1377, Nov 1990.

[38] S. Jain and V. Agrawal, "Statistical fault analysis," *Design Test of Computers, IEEE*, vol. 2, no. 1, pp. 38–44, Feb 1985.

[39] K. P. Parker and E. J. McCluskey, "Probabilistic treatment of general combinational networks," *Computers, IEEE Transactions on*, vol. 100, no. 6, pp. 668–670, 1975.

[40] J. Savir, G. S. Ditlow, and P. H. Bardell, "Random pattern testability," *Computers, IEEE Transactions on*, vol. 100, no. 1, pp. 79–90, 1984.

[41] S.-C. Chang, W.-B. Jone, and S.-S. Chang, "Tair: Testability analysis by implication reasoning," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 19, no. 1, pp. 152–160, 2000.

[42] K. Parker and E. McCluskey, "Analysis of logic circuits with faults using input signal probabilities," *Computers, IEEE Transactions on*, vol. C-24, no. 5, pp. 573–578, May 1975.

[43] J. F. Miller and S. L. Smith, "Redundancy and computational efficiency in cartesian genetic programming," *IEEE Trans. Evolutionary Computation*, vol. 10, no. 2, pp. 167–174, 2006.

[44] B. W. Goldman and W. F. Punch, "Analysis of cartesian genetic programming's evolutionary mechanisms," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 359–373, 2015.

[45] Z. Vasicek and L. Sekanina, "How to evolve complex combinational circuits from scratch?" in *2014 IEEE International Conference on Evolvable Systems Proceedings*. Institute of Electrical and Electronics Engineers, 2014, pp. 133–140. [Online]. Available: http://www.fit.vutbr.cz/research/view_pub.php?id=10673

[46] "Nangate freepdk15 open cell library," 2014. [Online]. Available: http://www.nangate.com/?page_id=2328

[47] H. K. Lee and D. S. Ha, "Hope: an efficient parallel fault simulator for synchronous sequential circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 15, no. 9, pp. 1048–1058, Sep 1996.

## A.7 EvoApprox8b: Library of Approximate Adders and Multipliers for Circuit Design and Benchmarking of Approximation Methods

Vojtech Mrazek, Radek Hrbacek, Zdenek Vasicek, Lukas Sekanina

In *Proc. of the 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*.
Lausanne: European Design and Automation Association, 2017.
ISBN 978-3-9815370-9-3, pp. 258-261.

# EvoApprox8b: Library of Approximate Adders and Multipliers for Circuit Design and Benchmarking of Approximation Methods

Vojtech Mrazek, Radek Hrbacek, Zdenek Vasicek and Lukas Sekanina
Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence
Brno, Czech Republic
Email: {imrazek, ihrbacek, vasicek, sekanina}@fit.vutbr.cz

***Abstract*—Approximate circuits and approximate circuit design methodologies attracted a significant attention of researchers as well as industry in recent years. In order to accelerate the approximate circuit and system design process and to support a fair benchmarking of circuit approximation methods, we propose a library of approximate adders and multipliers called EvoApprox8b. This library contains 430 non-dominated 8-bit approximate adders created from 13 conventional adders and 471 non-dominated 8-bit approximate multipliers created from 6 conventional multipliers. These implementations were evolved by a multi-objective Cartesian genetic programming. The EvoApprox8b library provides Verilog, Matlab and C models of all approximate circuits. In addition to standard circuit parameters, the error is given for seven different error metrics. The EvoApprox8b library is available at: www.fit.vutbr.cz/research/groups/ehw/approxlib**

## I. Introduction

Approximate circuits are becoming a viable alternative to conventional accurately operating circuits if energy efficiency is crucial and target application is error resilient [1]. This is the case of many, for example, image and video processing circuits that are predominantly composed of adders and multipliers. In order to approximate circuits such as adders and multipliers for a particular application, a designer can either perform a single purpose (ad hoc) approximation or apply some of the circuit approximation methods available in the literature. We will only deal with functional approximation in which logic function implemented by the original circuits is simplified. Other approximation techniques enabling power reduction such as voltage over scaling are not considered in this paper.

Unfortunately, almost all papers dealing with circuit approximation show some of the following features that are undesirable from a practical point of view: (1) The approximation method is described, but a corresponding software implementation is not available. (2) An implementation of the original (accurate) circuit is not available. (3) The quality of approximation and other parameters of approximate circuits are expressed relatively to parameters of the original circuit. If the original circuit is not available, it is hard or even impossible to obtain real parameters of the approximate circuits and reproduce the results. (4) Implementations of the resulting approximate circuits are not available. (5) Only a few approximate versions created from the original circuit are reported, forming thus a sparsely occupied Pareto front. (6) It is unclear if a given number of test vectors used to evaluate approximate circuits is sufficient for obtaining a trustworthy error quantification if the error is determined using simulation. (7) A given approximation method is only rarely compared against competitive approximation methods.

To the best of our knowledge, paper [2] and related software is the only one which does not suffer from the aforementioned problems. However, the paper is solely devoted to 16 bit adders and still only a few approximate adders can be downloaded.

The undesirable properties (1) – (7) are resulting in a situation in which the user does not know which approximation method is the most suitable one for his/her problem because the quality of available approximation methods cannot fairly be compared. If the user does not want to apply any of the approximation methods, its intention could be to use just a resulting approximate circuit showing desired parameter values. However, the chance of direct downloading a circuit with desired parameters is very low, because only a few approximate versions of the original circuit exist in the corresponding repository. On the other hand, it has to be emphasized that a fair comparison of approximation methods is difficult as they should be compared under the same conditions (under the same error metrics, fabrication technology, synthesis tools, available run-time etc.) and start with the same original circuit, which is difficult to ensure in the progressively developing field of approximate computing.

In order to address at least some of the aforementioned challenges, the objective of this paper is to introduce a rich and well-focused library of approximate components (called EvoApprox8b) that can be downloaded and immediately used in various applications or for benchmarking of circuit approximation methods. The library consists of approximate 8-bit adders and 8-bit multipliers that are crucial components of, for example, approximate image and video processing applications. In addition to circuit parameters (error, area, delay, power etc.), the library contains Matlab, C and Verilog implementations for all components which allows the user not only to immediately use the components, but also to calculate the error for a new target error metric or obtain desired parameters by performing a synthesis for a fabrication technology different to the reported one. In addition to that

978-3-9815370-8-6/17/$31.00 

these circuits can be utilized as building blocks of more complex approximate arithmetic circuits (as shown e.g. in [3]).

There are 430 different approximate 8-bit adders and 471 different approximate 8-bit multipliers forming a Pareto front in a three-dimensional space defined using the error, delay, and power metrics. These implementations were obtained by a multi-objective Cartesian genetic programming (CGP) according to [4]. As a starting point for the CGP-based approximation process, 13 different adder and 6 different multiplier accurately-operating architectures were employed. The adders include Ripple-Carry Adder (RCA), Carry-Select Adder (CSA), Carry-Look-ahead 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. These accurate implementations are also included in the library.

CGP has been adopted as our design method because papers [5], [6], [7] revealed that it can provide much better implementations of accurate as well as approximate circuits than common circuit design and optimization tools. The EvoApprox8b library is provided as an open source project, see the EvoApprox8b web site [8].

The rest of the paper is organized as follows. Section II surveys the methods developed for approximation of adders and multipliers. Section III describes the approximation method used to create the EvoApprox8b library. The library is presented in Section IV. Conclusions are given in Section V.

## II. Approximate adders and multipliers

Software-oriented benchmark sets such as AxBench [9] were developed for evaluation of approximate software and corresponding approximation methods. Regarding circuit approximation, several approximate circuits can be downloaded from KIT CES web site, including Generic Accuracy Configurable Adders (GeAR) [2]. The remaining papers dealing with circuit approximation methods suffer from problems discussed in Section I and, hence, performing a fair comparison of approximation methods or resulting approximate circuits is difficult. In this section, we will briefly survey approaches developed for adders and multipliers approximation.

Adders and multipliers are approximated by either general-purpose approximation methods or problem-specific methods. In the case of general-purpose methods (e.g. SALSA [10] and SASIMI [11]), adders and multipliers serve as one of many circuit classes that can be approximated. Problem-specific methods exploit the structure of conventional adders and multipliers. Another class of circuits are quality configurable circuits (e.g. GeAR adders) which allow for a dynamic approximation according the expected quality of result [2].

Four types of approximate adders are considered in [12]: (1) Speculative adders in which it is presumed that the probability that the carry propagation chain is longer than $k$ bits ($k << n$) is very low for $n$-bit adders. Hence, according to $k$ bits the carry is speculated for each sum bit. (2) Segmented adders, where an $n$-bit adder is divided into $k$-bit sub-adders and the carry is then generated by using different methods. (3) Carry-select adders in which multiple sub-modules are used to compute the sum for different carry values, and the result is determined according to the carry of a sub-module. (4) Approximate full adders where the full adder (an elementary adder's component) is approximated.

Approximate multipliers are based on the following principles [13]: (1) Approximation in generating partial products utilizing a simpler structure to generate partial products [3]. (2) Approximation in the partial product tree by ignoring some partial products or dividing partial products into several modules and applying approximation in the less significant modules. (3) Using approximate counters or compressors in the partial product tree. (4) Using approximate Booth multipliers. (5) Composing complex approximate multipliers by means of simple approximate multipliers as shown in [3].

Recently, an evolutionary approach was applied in the task of approximate circuits design with respect to multiple objectives [6], [7], [4]. Conventional circuits were used as an initial population. The circuit approximation problem is formulated as a multi-objective search problem and solved using the state-of-the art CGP method [14] combined with the NSGA-II algorithm [15]. In many cases, CGP-based approach is capable of optimizing accurate circuits in such a way that they remain accurate, but they show better parameters (e.g. area) than approximate circuits [4].

There are many error metrics developed to evaluate the quality of approximate arithmetic circuits [16]. While most methods apply test vectors to estimate the error (e.g. $10^8$ test vectors were used to evaluate 16 bit adders in [12]), the exact error calculation based on formal models such as binary decision diagrams is rarely used. The accuracy of simulation-based error calculation (which depends on the number and quality of test vectors) can significantly influence the whole approximation process.

## III. Circuit Approximation Method

The method used to obtain the library follows the methodology introduced in [4]. It is a general-purpose approximation method for combinational circuits based on a multi-objective CGP. It represents candidate circuits as directed acyclic graphs and tries to simultaneously minimize delay, power consumption and error to discover a set of approximate circuits along a Pareto front. Moreover, various constraints can be formulated to reduce the search space.

### A. Circuit representation

In CGP, candidate solutions are represented in a two-dimensional array of programmable nodes [14]. An $n_i$-input and $n_o$-output combinational circuit is modelled using an array of $n_c \cdot n_r$ programmable nodes forming a Cartesian grid. A set of available $n_a$-input/$n_b$-output node functions is denoted $\Gamma$. The primary inputs and programmable node outputs are uniquely numbered. For each node the chromosome contains ($n_a$+1) values that represent (i) node function and (ii) addresses specifying the input connections. The chromosome also contains $n_o$ values specifying the gates (node outputs)

connected to the primary outputs. The chromosome size is $n_c n_r (n_a + 1) + n_o$ integers.

In our case, $\Gamma$ contains functions implemented by standard components (such as gates and adders) of technology library. We used a subset of componets from a TSMC 180 nm technology library. A complete list of technology components including their area and leakage power can be found in [4]. There are multiple implementations of the same component differing in the implementation cost. During the circuit approximation, a proper size was selected for each gate depending on the output load of the gate.

### B. Search method

New candidate circuits are created by means of a point mutation operator which modifies a given number of integers of the chromosome. The multi-objective search is conducted by the NSGA-II algorithm which is based on the idea of *Pareto dominance relation*. The individuals in each generation are sorted according to the dominance relation into multiple fronts. The solutions within the individual fronts are sorted according to the *crowding distance* metric, which helps to preserve a reasonable diversity along the fronts [15]. The crowding distance is the average distance of two solutions on either side along each of the objectives. A new population is then created by exploiting appropriate Pareto fronts as defined in [17]. The result of a single evolutionary run is not just one solution, but a set of non-dominated solutions occupying the Pareto front. The search method is implemented as a parallel evolutionary algorithm operating with multiple populations distributed on several islands and the best individuals are allowed to migrate among the islands.

### C. Fitness functions

A selection of the error metric significantly influences obtained results [6]. As the error metric is usually an application-specific function there is no reason to prefer one over another in our library. We decided to optimize according to the mean relative error, but we also quantified the errors according to other commonly used error metrics for all evolved circuits (Section IV). The mean relative error is calculated accurately, i.e. for all possible $2^{n_i}$ input vectors, as:

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

where $O^{(i)}_{\mathrm{orig}}$ is the decimal representation of the $i$-th circuit correct output, $O^{(i)}_{\mathrm{approx}}$ is the individual's $i$-th output, and $n_i$ is the number of primary inputs (i.e. the operand's width is $n_i/2$ bits). In addition to that, we constrained the worst absolute and relative errors to reduce the search space.

The circuit area is a sum of the areas of components used in the circuit. Power consumption is estimated according to the algorithm given in [4]. The delay of a candidate circuit is calculated using the parameters defined in the liberty timing file available for the utilized semiconductor technology. The delay $t_{\mathrm{d}}$ of a cell $c_i$ is modeled as a function of its input transition time $t_{\mathrm{s}}$ and capacitive load $C_{\mathrm{l}}$ on the output of the cell, i.e. $t_{\mathrm{d}}(c_i) = f(t^{c_i}_{\mathrm{s}}, C^{c_i}_{\mathrm{l}})$. The delay of the circuit $C$ is determined as the delay along the longest path.

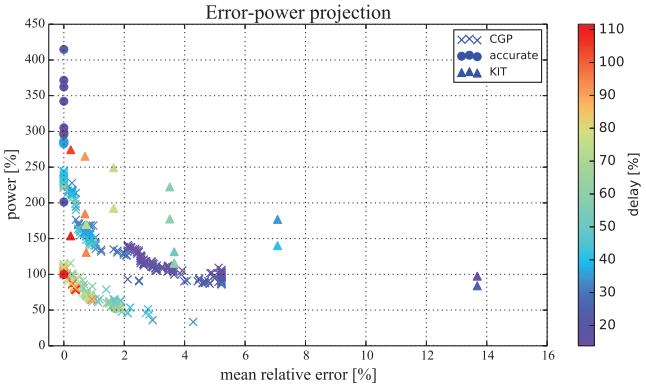



Fig. 1. Pareto fronts with parameters of evolved approximate 8-bit adders, conventional accurate adders and approximate adders according to KIT's paper [2].

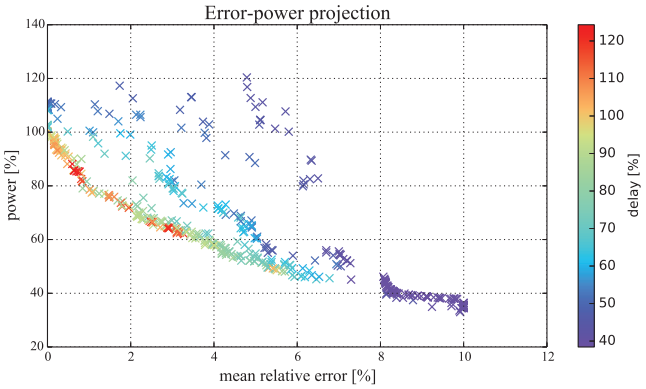



Fig. 2. Pareto fronts with parameters of evolved approximate 8-bit multipliers.

### D. Parameters setting

The CGP/NSGA-II parameters were set as follows: 500 individuals in the population, 100,000 generations, 10 islands, mutation rate 5 %, number of rows $n_{\mathrm{r}} = 1$. 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 %. All candidate solutions violating these requirements were discarded.

## IV. EvoApprox8b Library

The EvoApprox8b library contains 430 non-dominated 8-bit approximate adders evolved from the initial population of 13 conventional adders and 471 non-dominated 8-bit approximate multipliers that were evolved from 6 conventional implementations. In addition to the conventional implementations, the

library also contains 43 exact adders and 28 exact multipliers that were obtained by CGP and that are not dominated by any accurate implementation. All Pareto fronts are shown in Figure 1 and 2. All parameters are related to the Ripple-Carry Adder and Ripple-Carry Array Multiplier architectures (considered as 100% in the figures). Figure 1 also provides parameters of 8-bit approximate adders created according to [2]. Evolved adders show quite competitive properties under the metrics used in the figure.

All approximate circuits and their various parameters can be found at the EvoApprox8b web site [8]. It contains circuit models for Verilog, Matlab and C. This enables the user to integrate the circuits to hardware as well as software projects and design tools. All circuits can thus be simulated in order to obtain their other parameters that are not listed on the web site (e.g. errors under different error metrics or power consumption for another fabrication technology). The circuits can be sorted according to a chosen parameter which is useful when the user is looking for a circuit satisfying particular constraints.

The following list gives parameters that are provided for all circuits in the library: Area, delay, power (all estimated according to [4] which was validated against Cadence Encounter RTL Compiler and TSMC 180 nm library), nodes (the number of nodes, where a gate, a half adder and a full adder are counted as one node), HD (Hamming distance), EP (error probability), MAE (mean absolute error), MSE (mean squared error), MRE (mean relative error), WCE (worst case error), and WCRE (worst case relative error). Note that as $n_i$ is the number of primary inputs, the operand's width is $n_i/2$ bits.

$$\mathrm{HD} = \sum_{\forall i} OnesCount(O^{(i)}_{\mathrm{approx}} \oplus O^{(i)}_{\mathrm{orig}}), \tag{2}$$

$$\mathrm{EP} = \frac{\sum_{\forall i: O^{(i)}_{\mathrm{approx}} \neq O^{(i)}_{\mathrm{orig}}} 1}{2^{n_\mathrm{i}}}, \tag{3}$$

$$\mathrm{MAE} = \frac{\sum_{\forall i} \left| O^{(i)}_{\mathrm{approx}} - O^{(i)}_{\mathrm{orig}} \right|}{2^{n_\mathrm{i}}}, \tag{4}$$

$$\mathrm{MSE} = \frac{\sum_{\forall i} \left| O^{(i)}_{\mathrm{approx}} - O^{(i)}_{\mathrm{orig}} \right|^2}{2^{n_\mathrm{i}}}, \tag{5}$$

$$\mathrm{MRE} = \frac{\sum_{\forall i} \frac{\left| O^{(i)}_{\mathrm{approx}} - O^{(i)}_{\mathrm{orig}} \right|}{\max(1, O^{(i)}_{\mathrm{orig}})}}{2^{n_\mathrm{i}}}, \tag{6}$$

$$\mathrm{WCE} = \max_{\forall i} \left| O^{(i)}_{\mathrm{approx}} - O^{(i)}_{\mathrm{orig}} \right|, \tag{7}$$

$$\mathrm{WCRE} = \max_{\forall i} \frac{\left| O^{(i)}_{\mathrm{approx}} - O^{(i)}_{\mathrm{orig}} \right|}{\max(1, O^{(i)}_{\mathrm{orig}})}. \tag{8}$$

## V. Conclusions

In this paper we presented a rich library of approximate 8 bit adders and multipliers which is primarily intended for creating approximate blocks of image and video processing circuits. The Matlab, C and Verilog models that are available from [8] can allow software as well as hardware developers to integrate the approximate adders and multipliers to their designs. As we are aware of, this is the first open-source library containing hundreds of approximate components that allows for reproducible comparisons across various layers of design abstraction. These components can be utilized as building blocks of more complex approximate adders and multipliers as demonstrated for multipliers in [3].

## Acknowledgments

This work was supported by Czech science foundation project GA16-17538S and 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] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, pp. 86:1–86:6.

[3] 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.

[4] R. Hrbacek, V. Mrazek, and Z. Vasicek, "Automatic design of approximate circuits by means of multi-objective evolutionary algorithms," in *Proc.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, "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.

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

[7] ——, "Evolutionary approach to approximate digital circuits design," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 432–444, 2015.

[8] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina. (2016) Approximate 8-bit Adders and Multipliers. [Online]. Available: www.fit.vutbr.cz/research/groups/ehw/approxlib

[9] A. Yazdanbakhsh, D. Mahajan, P. Lotfi-Kamran, and H. Esmaeilzadeh, "Axbench: A multi-platform benchmark suite for approximate computing," *IEEE Design and Test*, 2016.

[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, 2013, pp. 1367–1372.

[12] H. Jiang, J. Han, and F. Lombardi, "A comparative review and evaluation of approximate adders," in *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*. ACM, 2015, pp. 343–348.

[13] H. Jiang, C. Liu, N. Maheshwari, F. Lombardi, and J. Han, "A comparative evaluation of approximate multipliers," in *IEEE/ACM International Symposium on Nanoscale Architectures*. IEEE, 2016, pp. 191–196.

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

[15] 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.

[16] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *IEEE Trans. Computers*, vol. 62, no. 9, pp. 1760–1771, 2013.

[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.