A SAT-based Fitness Function for Evolutionary Optimization of Polymorphic Circuits

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

***Abstract*—Multifunctional (or polymorphic) gates have been utilized as building blocks for multifunctional circuits that are capable of performing various logic functions under different settings of control signals. In order to effectively synthesize polymorphic circuits, several methods have been developed in the recent years. Unfortunately, the methods are applicable for small circuits only. In this paper, we propose a SAT-based functional equivalence checking algorithm to eliminate the fitness evaluation time which is the most critical overhead for genetic programming-based design and optimization of complex polymorphic circuits. The proposed approach has led to a 20%-40% reduction in gate count with respect to the solutions created using the polymorphic multiplexing.**

I. Introduction

Recent research has led to a new class of reconfigurable components, so-called multifunctional or polymorphic gates, where the multi-functionality is achieved by a unique physical design of those components. While polymorphic gates were described by the NASA JPL evolvable hardware group in 2001 [1], graphene reconfigurable logic devices, discovered at IBM T. J. Watson Research Center, were presented in 2010 [2]. As this new technology does not utilize an expensive reconfiguration infrastructure (such as switches, multiplexers, configuration registers etc.), it enables to obtain reconfigurable and thus potentially adaptive circuits for a very low cost. Various external stimuli (such as standard Boolean signals, (multiple) external voltages, V_{dd} or temperature) have been investigated to control the behavior of multifunctional gates. For example, Stoica's polymorphic bifunctional NAND/NOR gate controlled by V_{dd} operates as NOR for V_{dd} = 1.8 V and NAND for V_{dd} = 3.3 V [3].

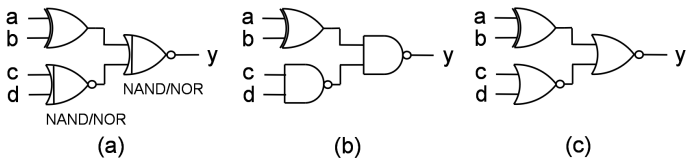

Fig. 1. A polymorphic circuit (a) and its equivalent scheme in the first mode (b) and the second mode (c)

Figure 1a shows a polymorphic circuit containing two polymorphic NAND/NOR gates and an ordinary XOR gate. Its corresponding schemes are shown when the NAND function is active (b) and the NOR function is active (c). Such circuit has two modes of operations. In general, k ($k > 1$) modes may be available. In this work, we assume that

- all polymorphic gates included in a circuit are controlled using the same control signal and
- ordinary gates perform identically in all modes.

The goal of polymorphic circuit synthesis can be formulated as a problem of finding such a circuit network which performs required functions $f_1 \dots f_k$ in modes $1 \dots k$ of polymorphic gates [4], minimizes various criteria (area, delay, power consumption etc.) and satisfies given constraints.

A straightforward approach to synthesis of polymorphic circuits is polymorphic multiplexing (see Section II-A). This method produces large and far-from-optimum solutions if the number of gates is the optimization criterion. Evolutionary design and evolutionary optimization methods have been employed to reduce the number of gates. However, they are not scalable as they are capable of producing some results for circuits with up to 8-15 inputs only (depending on a particular instance) [5].

In this paper, we propose a SAT-based functional equivalence checking algorithm to eliminate the main bottleneck of evolutionary optimization of polymorphic circuits. The algorithm has been embedded into the Cartesian genetic programming (CGP) method. We will show that it is possible to effectively optimize polymorphic circuits with tens of inputs. For experiments we will utilize the LGSynth93 benchmark set. A comparison will be performed against the polymorphic multiplexing where the initial solutions are created using the ABC tool.

The rest of the paper is organized as follows. Polymorphic circuits are introduced in Section II. Section III presents CGP as a method for evolution of ordinary as well as polymorphic circuits. The proposed SAT-based equivalence checking algorithm for polymorphic circuits is described in Section IV. The results of experiments are summarized and discussed in Section V. Finally, conclusions are given in Section VI.

II. Polymorphic Circuits

The behavior of polymorphic gates can be controlled either by a specialized signal (Boolean signal or voltage signal), power supply voltage or environment (temperature). Description of particular polymorphic gates and their static and dynamic characteristics can be found in [3], [2], [6]. Applications

978-3-9810801-8-6/DATE12/©2012 EDAA

of polymorphic electronics fall to the areas of intelligent filters, multifunctional counters, self-checking circuits and circuit testing. In addition to a few works on theoretical foundations of polymorphic circuits [7], [8], various methods have been proposed to design polymorphic circuits at the gate level.

A. Polymorphic Multiplexing

Consider an m-input/n-output polymorphic circuit operating in two modes only and implementing functions F_1 and F_2 (both with m-inputs and n-outputs). One can synthesize conventional circuits for F_1 and F_2 using a conventional synthesis tool, e.g. ABC [9]. In polymorphic multiplexing, we just connect the corresponding outputs of F_1 and F_2 by polymorphic multiplexers [5]. The structure of a polymorphic multiplexer is shown in Figure 2. The given polymorphic multiplexer propagates signal a in the first mode of polymorphic gates and signal b in the second mode of polymorphic gates. In order to reduce the number of gates, the goal of synthesis can be to maximize the number of gates that are shared by both circuits and minimize the number of outputs that have to be equipped with polymorphic multiplexers.

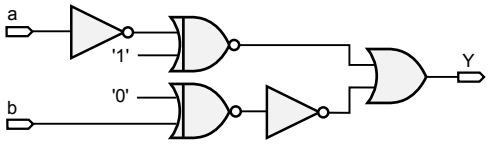

Fig. 2. A polymorphic multiplexer using NAND/NOR gates

B. Polymorphic BDDs

Polymorphic binary decision diagrams (PolyBDDs) were introduced as an alternative to polymorphic multiplexing [5]. PolyBDDs extend ordinary binary decision diagrams in such a way that terminal nodes can contain polymorphic logic functions. Decision nodes are then implemented using standard multiplexers.

C. Evolutionary Approaches

Evolutionary algorithms have been utilized in the phase of design as well as optimization of combinational polymorphic circuits. In the case of evolutionary design, Cartesian genetic programming (see next section) allows producing of complete polymorphic networks [4], [10]. However, the method is applicable for small problem instances only. In the case of optimization, CGP can be seeded by a circuit created using polymorphic multiplexing or PolyBDDs and used to reduce the number of gates. Again, a scalability limit is inherently present and the optimization does not produce reasonable results for circuits containing more than approx. 15 inputs [5].

III. Direct Evolution of Polymorphic Circuits Using CGP

Cartesian Genetic Programming has been developed for circuit evolution by Miller and Thompson [11], [12].

From the perspective of circuit design, a candidate circuit is represented as an array of n_c (columns) $\times$ n_r (rows) of programmable gates. The number of inputs, n_i, and outputs, n_o, is fixed. Each gate can be connected either to the output of a gate placed in previous l columns or to one of the circuit inputs. Setting of the l parameter allows to control the maximum circuit delay. Feedback is not allowed. Each gate has a constant number of inputs n_a and is programmed to perform one of functions defined in the set Γ. The number of inputs n_a is fixed, usually $n_a = 2$. If the arity a of a particular function is lower than the number of inputs n_a, only the first a inputs are involved in calculation.

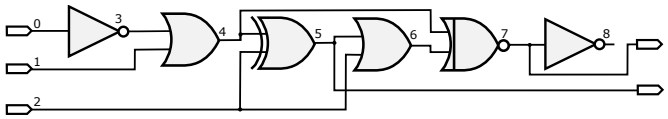

Fig. 3. Example of a candidate circuit encoded using CGP with the following parameters: $n_c = 6$, $n_r = 1$, $n_i = 3$, $n_o = 2$, $l = 2$, $n_a = 2$, Γ = {BUF (0), AND (1), OR (2), XOR (3), NOT (4), NAND/NOR (5)}. Chromosome: 0, 1, **4**, 3, 1, **1**, 4, 2, **3**, 5, 2, **1**, 4, 6, **5**, 7, 2, **4**, 7, 5. Function of each node is typed in bold. The first 18 integers (i.e. six triplets) encode the interconnection of the CGP nodes and logic function of each element. The last two integers indicate the output of the circuit. Node 8 is not utilized. Note that node 3 and 8 use only first and last integer of the corresponding triplet.

Each gate is encoded using $n_a + 1$ integers where values $1 \ldots n_a$ are the indexes of the input connections and the last value is the function code. The primary inputs are addressed by indexes $0, 1 \ldots n_i - 1$ and programmable gates by $n_i \ldots n_c.n_r + n_i - 1$. Every circuit is encoded using $n_c.n_r.(n_a + 1) + n_o$ integers. The last n_o integers define the primary outputs of the circuit. In the case of polymorphic circuit evolution, Γ also contains polymorphic gates. Figure 3 shows an example of a candidate circuit and its chromosome.

CGP operates with a population of $1 + \lambda$ individuals (typically, $\lambda = 4$ [13]). The initial population is constructed either randomly or seeded by a conventional design. Every new population consists of the best individual of the previous population and its λ offspring individuals. The offspring individuals are created using a point mutation operator which modifies h randomly selected genes of the chromosome, where h is a user-defined value. This search strategy has been found as the most useful in literature [13].

In case of evolutionary design of ordinary combinational circuits, the fitness value of a candidate circuit is defined as:

$$fit1 = \begin{cases} b & when \quad b < n_o 2^{n_i}, \\ b + (n_c n_r - z) & otherwise, \end{cases} \tag{1}$$

where b is the number of correct output bits obtained as response for all possible assignments to the inputs, z denotes the number of gates utilized in a particular candidate circuit and $n_c.n_r$ is the total number of available gates. It can be seen that the last term $n_c n_r - z$ is considered only if the circuit behavior is perfect, i.e. $b = b_{max} = n_o 2^{n_i}$. Then, the number of gates is optimized. Similarly, delay or power consumption can be optimized. Unfortunately, the evaluation time grows exponentially with respect to the number of primary inputs.

The extension of the fitness function for evaluation of polymorphic circuits is straightforward. Let us assume that $k = 2$

(i.e., polymorphic circuits have two modes of operations) and the number of gates has to be minimized. A candidate circuit is set to the first mode and the number of correct bits (b_1) is calculated with respect to the first target function F_1. Then, the circuit is set to the second mode and b_2 is calculated for F_2. After reaching a perfect functionality in both modes, the number of gates can be optimized:

$$fit2 = \begin{cases} b_1 + b_2 & when \quad b_1 + b_2 < 2n_o2^{n_i}, \\ b_1 + b_2 + (n_c n_r - z) & otherwise, \end{cases} \tag{2}$$

It is evident again that this method is not scalable.

IV. Proposed Fitness Evaluation for Polymorphic Circuits

In order to accelerate the evaluation of candidate circuits, a SAT-based approach has been introduced [14], [15]. We propose to extend this approach for polymorphic circuits. The goal is to minimize the number of gates.

A. The SAT-based Equivalence Checking

It is assumed that CGP is seeded using a fully functional polymorphic circuit which is obtained from polymorphic multiplexing. The seed is utilized as a *reference* solution for a SAT-based equivalence checking algorithm, which works as follows.

1) The reference circuit (A) as well as candidate circuit (B) is set to the first mode.
2) A new auxiliary circuit C1 is composed of the reference circuit in the first mode (circuit A1), the candidate circuit in the first mode (circuit B1) and a miter (a set of XOR gates followed by the OR detector). Circuit C1 is transformed into one Boolean formula in conjunctive normal form (CNF) which is unsatisfiable if and only if circuits A1 and B1 are functionally equivalent [16]. The transformation to CNF is conducted gate by gate using the Tseitin's algorithm [17]. Table I contains the CNF representation for selected gates. The resulting CNF is submitted to the MiniSAT solver [18].
3) If A1 and B1 are not functionally equivalent then the fitness evaluation is finished and CGP proceeds with another candidate circuit. Go to step (8).
4) As the reference circuit and candidate circuit are functionally equivalent in the first mode, the second mode can be investigated. Circuits A and B are set to the second mode.

TABLE I
CNF representation of some common gates

Gate	Corresponding CNF representation
$y = \mathrm{NOT}(x_1)$	$(\neg y \vee \neg x_1) \wedge (y \vee x_1)$
$y = \mathrm{AND}(x_1, x_2)$	$(y \vee \neg x_1 \vee \neg x_2) \wedge (\neg y \vee x_1) \wedge (\neg y \vee x_2)$
$y = \mathrm{OR}(x_1, x_2)$	$(\neg y \vee x_1 \vee x_2) \wedge (y \vee \neg x_1) \wedge (y \vee \neg x_2)$
$y = \mathrm{XOR}(x_1, x_2)$	$(\neg y \vee \neg x_1 \vee \neg x_2) \wedge (\neg y \vee x_1 \vee x_2) \wedge$ $(y \vee \neg x_1 \vee x_2) \wedge (y \vee x_1 \vee \neg x_2)$
$y = \mathrm{NAND}(x_1, x_2)$	$(\neg y \vee \neg x_1 \vee \neg x_2) \wedge (y \vee x_1) \wedge (y \vee x_2)$
$y = \mathrm{NOR}(x_1, x_2)$	$(y \vee x_1 \vee x_2) \wedge (\neg y \vee \neg x_1) \wedge (\neg y \vee \neg x_2)$

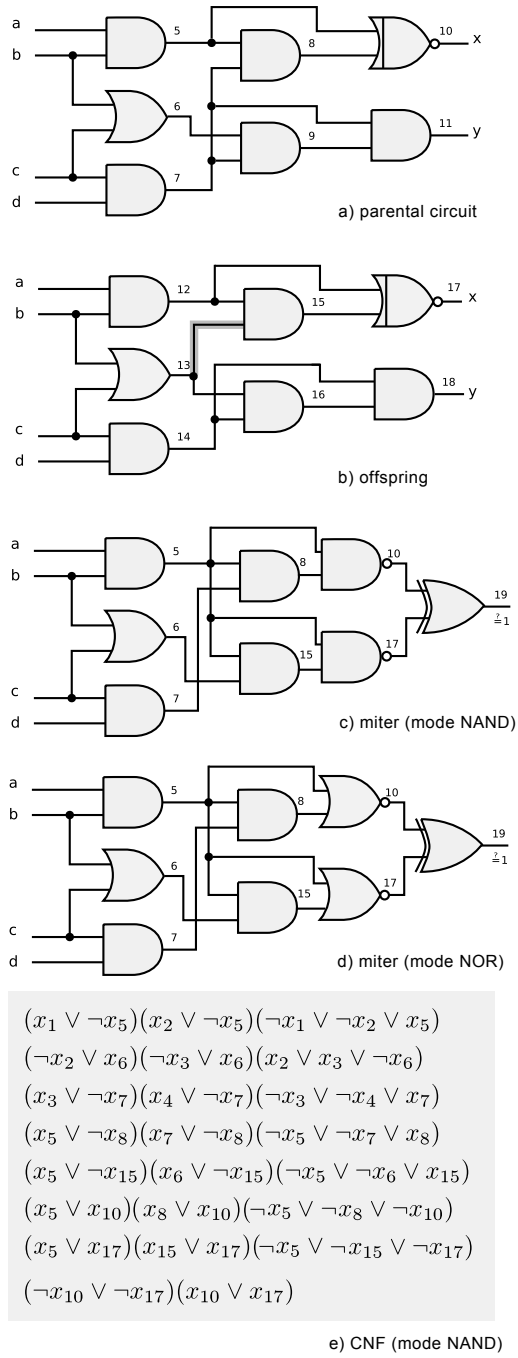

Fig. 4. Example of reference circuit (a) and its offspring (b) before a conversion to CNF is performed. A 'diference' circuit for the first mode (c) is constructed and transformed to CNF (e). If the CNF is unsatisfiable, a difference circuit for the second mode (d) is created and transformed to CNF. If the CNF is also unsatisfiable, the offspring is functionally equivalent to its parental circuit.

5) A new auxiliary circuit C2 is composed of the reference circuit in the second mode (circuit A2), the candidate circuit in the second mode (circuit B2) and a miter. Circuit C2 is transformed into one Boolean formula in CNF using the same procedure as described in step (2) and the obtained formula is submitted to the MiniSAT solver.

TABLE II
BENCHMARK CIRCUITS

circuit	circuit1	n_i	n_o	N_g	circuit2	n_i	n_o	N_g	seed1	seed2	N_p
C01	apex7.blif	49	37	208	cht.blif	47	36	156	544	534	180
C02	c8.blif	28	18	145	lal.blif	26	19	88	323	291	90
C03	c8.blif	28	18	145	misex2.blif	25	18	114	349	311	90
C04	c8.blif	28	18	145	pcler8.blif	27	17	91	321	286	85
C05	count.blif	35	16	130	my_adder.blif	33	17	131	341	306	80
C06	lal.blif	26	19	88	misex2.blif	25	18	114	292	267	90
C07	lal.blif	26	19	88	ttt2.blif	24	21	160	343	315	95
C08	misex2.blif	25	18	114	ttt2.blif	24	21	160	364	332	90
C09	pcler8.blif	27	17	91	lal.blif	26	19	88	264	249	85
C10	pcler8.blif	27	17	91	misex2.blif	25	18	114	290	272	85
C11	seq.blif	41	35	1927	C1355.blif	41	32	238	2325	2256	160
C12	seq.blif	41	35	1927	C499.blif	41	32	198	2285	2258	160
C13	unreg.blif	36	16	100	count.blif	35	16	130	310	310	80
C14	unreg.blif	36	16	100	my_adder.blif	33	17	131	311	307	80
C15	vda.blif	17	39	749	pdc.blif	16	40	871	1815	1575	195

6) If A2 and B2 are not functionally equivalent then the fitness evaluation is finished and CGP proceeds with another candidate circuit. Go to step (8).
7) Circuits A and B are functionally equivalent in both modes. The fitness value is given by the number of gates.
8) The end of fitness evaluation.

B. Further Acceleration

Additional acceleration can be obtained when the CNF is constructed in a smart way. One can utilize knowledge of genes which have been modified by the mutation operator to calculate a 'difference' between the parent individual and its offspring. Note that this 'difference' circuit is sufficient for checking the functional equivalence of parent circuit and its offspring and thus only the 'difference' is submitted to the SAT solver. An example of a reference (parent) circuit and modified circuit (offspring) is shown in Figure 4a,b. The 'difference' circuit for the first and second mode (Figure 4c,d) consists of 8 gates (7 + 1 XOR). This is a significant reduction with respect to the standard all-output approach which led to 17 gates (14 + 2 XOR + 1 OR). Resulting CNF for the first mode is shown in Figure 4e [15].

V. RESULTS

A. Benchmark Set

In order to evaluate the proposed method, we established a set of 15 polymorphic circuits C01-C15 (see Table II). A benchmark circuit C is constructed as follows: Two circuits, A (with n_i^A inputs and n_o^A outputs) and B (with n_i^B inputs and n_o^B outputs), were chosen from the LGSynth91 set such that they have a similar number of inputs and outputs. Then, $n_i^C = max(n_i^A, n_i^B)$ and $n_o^C = max(n_o^A, n_o^B)$. The outputs $1 \ldots min(n_o^A, n_o^B)$ of A and B are connected using polymorphic multiplexers. Remaining outputs are not polymorphic. Similarly, the inputs $1 \ldots min(n_i^A, n_i^B)$ are directly connected, remaining inputs are not shared. The number of gates of an initial ordinary circuit is N_g. The number of gates of a polymorphic circuit is $N_g^A + N_g^B + c.min(n_o^A, n_o^B)$ where $c = 5$ is (according to Figure 2) the number of gates of a polymorphic multiplexer (see the seed1 column in Table II).

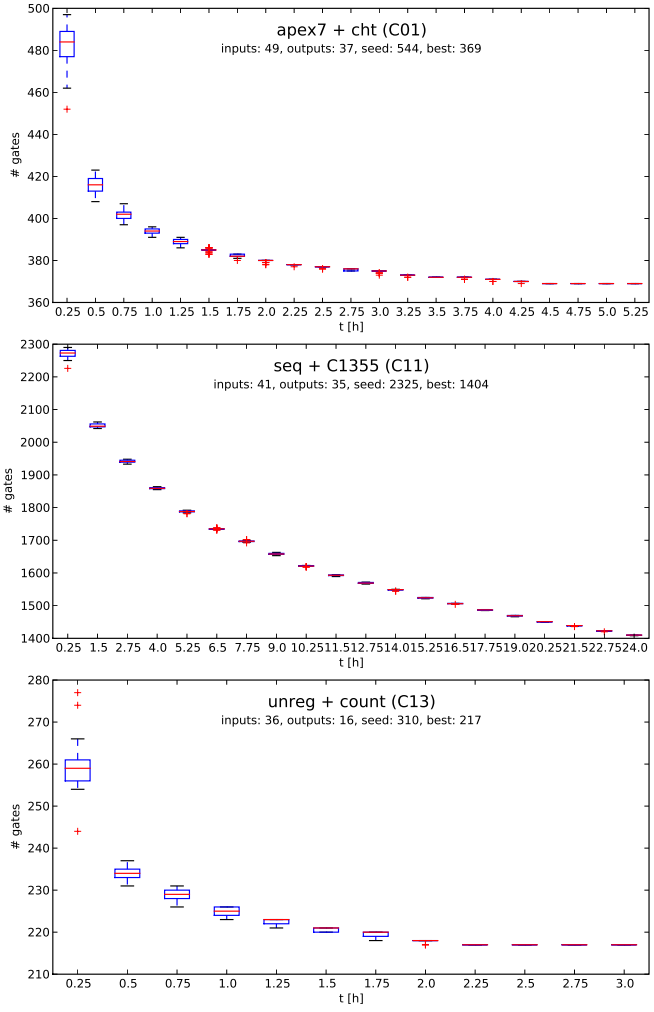

Fig. 5. The progress of optimization for selected circuits: Box plots from 25 independent runs of the CGP optimizer

N_p is the total number of gates consumed by polymorphic multiplexers. The seed2 column gives the number of gates of an optimized polymorphic circuit which was created after merging and optimizing A and B using the ABC tool. This optimization has led to a small area reduction for most circuits.

TABLE III
THE RESULTING NUMBER OF GATES (N_b – TOTAL, N_n – POLYMORPHIC) FOR CGP SEEDED BY SEED1

	seed		1h runtime		4h runtime		best				
circuit	N_b	N_n	N_b	N_n	N_b	N_n	N_b	N_n	impr.	p_{chg}	cmpeff
C01	544	72	391	43	373	48	369	44	32.2%	-38.9%	7.7
C02	323	36	214	36	201	39	189	33	41.5%	-8.3%	10.7
C03	349	36	234	51	218	56	210	52	39.8%	44.4%	7.5
C04	321	34	214	51	204	53	202	50	37.1%	47.1%	8.8
C05	341	32	258	37	235	41	227	43	33.4%	34.4%	3.1
C06	292	36	174	28	163	25	163	28	44.2%	-22.2%	11.3
C07	343	38	193	26	183	29	183	30	46.6%	-21.1%	9.0
C08	364	36	214	35	207	35	206	31	43.4%	-13.9%	8.6
C09	264	34	160	34	153	37	151	32	42.8%	-5.9%	11.3
C10	290	34	178	45	171	41	165	45	43.1%	32.4%	7.5
C11	2325	64	2093	141	1920	141	1404	118	39.6%	84.4%	0.3
C12	2285	64	2061	135	1853	115	1353	113	40.8%	76.6%	0.3
C13	310	32	223	40	217	36	217	36	30.0%	12.5%	8.2
C14	311	32	236	34	221	33	220	32	29.3%	0.0%	5.1
C15	1815	78	1470	148	1266	137	885	113	51.2%	44.9%	0.6

B. CGP Setup

We applied CGP to minimize the number of gates in the benchmark circuits. Another goal was to observe the effect of seeding the initial population by seed1 and seed2. The CGP parameters were initialized according to suggestions given in [14], [15]. The CGP array contains $n_c \times 1$ nodes, where n_c is the number of gates in the seed circuit, $\lambda = 1$, $l = n_c$ and $h = 2$. The set of functions contains ordinary two-input logic functions, buffer, inverter and the NAND/NOR gate, i.e. $\Gamma = \{\mathrm{BUF}, \mathrm{NOT}, \mathrm{AND}, \mathrm{OR}, \mathrm{XOR}, \mathrm{NAND}, \mathrm{NOR}, \mathrm{NAND/NOR}, \mathrm{ZERO}, \mathrm{ONE}\}$. Similarly to [5], we have used just one polymorphic function. Note that the constant generators (ZERO and ONE) have been utilized to create the polymorphic multiplexers only, i.e. the mutation operator has not generated any nodes having such a function. It is also assumed that the implementation cost of all gates is identical.

Similarly to [14], the MiniSAT 2 (version 070721) has been used as a SAT solver [18] because it can easily be embedded into a custom application. The experiments were carried out on a cluster consisting of Intel Xeon E5345 2.33 GHz processors that enables to run several experiments in parallel.

C. Experiments

Table III shows the minimum number of gates (N_b) that were obtained for CGP seeded by seed1. We have used 25 independent evolutionary optimizers running in parallel. The best-current individual was identified every 15 minutes and CGP was reseeded by it. In order to compare the convergence speed, Table III contains the number of gates after 1 hour and 4 hours of the optimization running on a 2.4 GHz processor. It also contains the best result (a minimum circuit) which has been extracted when the progress of the optimization stagnated for three consecutive 15-minute intervals. We can observe that the number of gates was reduced by 35% in average with respect to polymorphic multiplexing.

N_n denotes the number of polymorphic NAND/NOR gates in resulting circuits. The p_{chg} column gives how the number of polymorphic gates increased/decreased in comparison to the initial circuit. Intuitively, compact polymorphic circuits are characterized by a high number of polymorphic gates. However, as it can be seen, some of the optimized circuits exhibit significantly lower number of polymorphic gates compared to initial seed (e.g. C01). These results indicate that the initial solutions probably contain a certain degree of redundancy; i.e., these circuits can be represented in a more compact way.

Another set of experiments was performed for CGP seeded using seed2. Table IV shows that although seed2 represents a smaller circuit than seed1, we obtained more compact circuits only in a few cases in comparison to CGP seeded by seed1. The 'cmpeff' column gives the number of evaluations (in millions) per hour.

Figures 5 shows statistical results collected during the evolution for selected benchmark circuits. The box plots were calculated using 25 independent runs.

VI. CONCLUSIONS

In this paper, we proposed a new CGP-based method for evolutionary optimization of polymorphic circuits. The method is capable of reducing the number of gates for circuits containing tens of inputs which was unreachable using former approaches.

The initial polymorphic circuits (the best results from the ABC-based polymorphic multiplexing) were significantly improved by CGP. This confirms the conclusion of papers [14], [15] that CGP can reduce the number of gates by 20-40% in comparison to conventional methods. One can also observe that independent runs led to very similar results at the end of evolution which indicates that CGP exhibits a very stable behavior. We also showed that the role of the seed in not crucial. It usually helps only insignificantly to resynthesize and optimize the polymorphic circuit using ABC before the polymorphic mutiplexers are introduced. Although CGP quickly converges at the beginning of evolution, it may take a considerable time to terminate the calculations. In general, good results are obtained for the cost of runtime.

Our future work will be devoted to improving the SAT based equivalence checking algorithm for polymorphic circuits as well as the CGP optimizer.

TABLE IV
THE RESULTING NUMBER OF GATES (N_b – TOTAL, N_n – POLYMORPHIC) FOR CGP SEEDED BY SEED2

	seed		1h runtime		4h runtime		best				
circuit	N_b	N_n	N_b	N_n	N_b	N_n	N_b	N_n	impr.	p_{chg}	cmpeff
C01	534	72	412	48	395	60	375	63	29.8%	-12.5%	16.2
C02	291	36	203	46	189	51	189	47	35.1%	30.6%	5.5
C03	311	36	230	53	216	56	201	58	35.4%	61.1%	10.9
C04	286	34	201	52	194	53	190	51	33.6%	50.0%	11.6
C05	306	32	264	41	249	49	244	49	20.3%	53.1%	3.0
C06	267	36	172	34	163	34	159	31	40.4%	-13.9%	12.2
C07	320	38	189	35	180	33	179	32	44.1%	-15.8%	7.8
C08	337	36	222	33	215	32	215	36	36.2%	0.0%	7.2
C09	254	34	159	28	157	29	157	29	38.2%	-14.7%	11.3
C10	272	34	176	36	174	34	174	34	36.0%	0.0%	8.5
C11	2261	64	2050	106	1844	103	1368	92	39.5%	43.8%	0.3
C12	2263	64	2052	111	1851	105	1399	100	38.2%	56.3%	0.3
C13	310	32	238	51	230	57	226	52	27.1%	62.5%	7.6
C14	307	32	252	45	240	49	236	44	23.1%	37.5%	4.8
C15	1575	78	1224	99	1092	91	888	71	43.6%	-9.0%	0.5

VII. ACKNOWLEDGMENTS

This work was supported by the Czech science foundation project P103/10/1517, the research programme MSM 0021630528 and the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070.

REFERENCES

[1] A. Stoica, R. S. Zebulum, and D. Keymeulen, "Polymorphic electronics," in *Proc. of Evolvable Systems: From Biology to Hardware Conference*, ser. LNCS, vol. 2210. Springer, 2001, pp. 291–302.

[2] S. Tanachutiwat, J. U. Lee, W. Wang, and C. Y. Sung, "Reconfigurable multi-function logic based on graphene p-n junctions," in *Design Automation Conference, DAC*. ACM, 2010, pp. 883–888.

[3] A. Stoica, R. Zebulum, X. Guo, D. Keymeulen, I. Ferguson, and V. Duong, "Taking Evolutionary Circuit Design From Experimentation to Implementation: Some Useful Techniques and a Silicon Demonstration," *IEE Proc.-Comp. Digit. Tech.*, vol. 151, no. 4, pp. 295–300, 2004.

[4] L. Sekanina, "Evolutionary design of gate-level polymorphic digital circuits," in *Applications of Evolutionary Computing*, ser. LNCS, vol. 3449. Lausanne, Switzerland: Springer Verlag, 2005, pp. 185–194.

[5] Z. Gajda and L. Sekanina, "On evolutionary synthesis of compact polymorphic combinational circuits," *Journal of Multiple-Valued Logic and Soft Computing*, vol. 17, no. 6, pp. 607–631, 2011.

[6] V. Simek, R. Ruzicka, and L. Sekanina, "On analysis of fabricated polymorphic circuits," in *Proc. of the 13th Int. IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*. IEEE Computer Society, 2010, pp. 281–284.

[7] W. Luo, Z. Zhang, and X. Wang, "Designing polymorphic circuits with polymorphic gates: a general design approach," *IET Circuits, Devices & Systems*, vol. 1, no. 6, pp. 470–476, 2007.

[8] Z. Li, W. Luo, L. Yue, and X. Wang, "On the completeness of the polymorphic gate set," *ACM Transactions on Design Automation of Electronic Systems*, vol. 15, no. 4, p. 25, 2011.

[9] Berkley Logic Synthesis and Verification Group, *ABC: A System for Sequential Synthesis and verification*. [Online]. Available: http://www.eecs.berkeley.edu/˜ alanmi/abc/

[10] H. Liang, W. Luo, and X. Wang, "Designing polymorphic circuits with evolutionary algorithm based on weighted sum method," in *Evolvable Systems: From Biology to Hardware*, ser. LNCS, vol. 4684. Springer, 2007, pp. 331–342.

[11] J. F. Miller and P. Thomson, "Cartesian Genetic Programming," in *Proc. of the 3rd European Conference on Genetic Programming EuroGP2000*, ser. LNCS, vol. 1802. Springer, 2000, pp. 121–132.

[12] J. F. Miller, D. Job, and V. K. Vassilev, "Principles in the Evolutionary Design of Digital Circuits – Part I," *Genetic Programming and Evolvable Machines*, vol. 1, no. 1, pp. 8–35, 2000.

[13] J. F. Miller and S. L. Smith, "Redundancy and Computational Efficiency in Cartesian Genetic Programming," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 2, pp. 167–174, 2006.

[14] 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.

[15] ——, "A global postsynthesis optimization method for combinational circuits," in *Proc. of the Design, Automation and Test in Europe, DATE*. EDAA, 2011, pp. 1525–1528.

[16] E. Goldberg, M. Prasad, and R. Brayton, "Using SAT for combinational equivalence checking," in *DATE '01: Proceedings of the conference on Design, automation and test in Europe*. Piscataway, NJ, USA: IEEE Press, 2001, pp. 114–121.

[17] G. S. Tseitin, "On the complexity of derivation in propositional calculus," in *Studies in Constructive Mathematics and Mathematical Logic, Part II*, 1968, pp. 115–125.

[18] N. Een and N. Sorensson, "MiniSAT." [Online]. Available: http://minisat.se