Parallel Optimization of Transistor Level Circuits using Cartesian Genetic Programming

Vojtech Mrazek
Faculty of Information Technology, Centre of Excellence IT4Innovations
Brno University of Technology
Brno, Czech Republic 612 66
imrazek@fit.vutbr.cz

Zdenek Vasicek
Faculty of Information Technology, Centre of Excellence IT4Innovations
Brno University of Technology
Brno, Czech Republic 612 66
vasicek@fit.vutbr.cz

ABSTRACT

The aim of the paper is to introduce a new parallel approach to evolutionary optimization of digital circuits described on transistor level. The evolutionary optimization is guided by the fitness function employing a simulator of candidate circuits. A new discrete simulator was introduced to achieve a good trade-off between precision and cost of circuit evaluations. The simulator is based on event-driven simulation. Precise numeric SPICE simulator is regularly called to validate simulation results. To increase the speed of evolution, three parallel approaches were proposed: (i) thread level parallelism, (ii) multiple computing nodes which collectively communicate and distribute the best solution, and (iii) client-server architecture eliminating a limited count of SPICE simulator instances.

CCS CONCEPTS

•Computing methodologies → Parallel algorithms; Search methodologies; •Hardware → *Power estimation and optimization;*

KEYWORDS

Evolutionary optimization, transistor-level, parallel systems, digital circuits

ACM Reference format:
Vojtech Mrazek and Zdenek Vasicek. 2017. Parallel Optimization of Transistor Level Circuits using Cartesian Genetic Programming. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3067695.3084212

1 INTRODUCTION

In recent years, many of papers showing the advantages of evolutionary design techniques in the field of digital circuit design have been published. Efficient implementations of various combinational circuits better than conventional design approaches can provide have been obtained by using Cartesian genetic programming (CGP). This branch of GP currently represents probably most effective technique in evolutionary logical circuits design [6–8, 15].

However, while the gate-level evolutionary synthesis represents an intensively studied research area, the synthesis of transistor-level digital circuits remains, in contrast with design of transistor-level analog circuits, on a peripheral concern of the researchers. Only a few papers were devoted to evolution of digital circuits directly at the transistor level. One of the first experiments of evolutionary design analog and discrete circuits were done by Vellasco et al. [16]. Zaloudek et al. published an approach based on a simple circuit simulator which was designed for rapid evaluation of candidate solutions [18]. Trefzer used another technique to evolve some basic logic gates [14]. Instead of using a time consuming analogue circuit simulator, a reconfigurable analog transistor array was employed. However, it was shown that the performance of many solutions decreased in simulation for various technologies; about 50% of discovered circuits failed in the simulation. The rough approximation of transistor behavior used in fitness function unfortunately caused that this approach tended to produce incorrectly working circuits. Walker et al. adopted detailed circuit simulation in the fitness function [17]. There was a problem with enormous computational overhead introduced by SPICE-based simulator. For that reason, it was possible to evolve correct solutions for small problem instances only.

Evolutionary algorithms are good candidates for parallel implementation. The parallelism can be introduced on various levels. In distributed evolutionary algorithms, best individuals are exchanged among independently evolving population (islands). In addition to the best individuals, probabilistic models and other parameters can be exchanged in these distributed approaches [2]. Another approach is to parallelize the fitness calculation, because it is the most time consuming procedure in real world applications. In the field of digital evolution, GPUs [5], Intel Xeon accelerators [3] or FPGAs [1] were used.

The goal of the paper is to compare three different approaches for parallel evolutionary circuit design which combines low-cost simulation and precise circuit simulation by means of SPICE.

2 EVOLUTIONARY DESIGN OF TRANSISTOR-LEVEL CIRCUITS

In order to evolve complex digital circuits on transistor-level, a suitable circuit representation enabling to encode bidirectional graph structures containing junctions is needed. To address this problem, we utilized encoding that is based on CGP [6] as proposed in [12].

GECCO '17 Companion, Berlin, Germany

DOI: http://dx.doi.org/10.1145/3067695.3084212

2.1 Circuit Representation

Each candidate solution is represented by means of an array of elementary nodes arranged in n_c columns and n_r rows. Each node consists of one output pin and two input pins. The input pins can independently be connected to a) the output pin of any node in previous columns, b) the circuit primary input. According to its configuration, each node can act as pmos transistor, nmos transistor, or junction of wires.

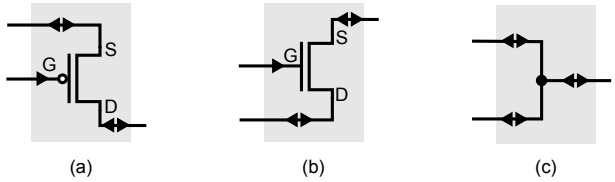

Figure 1: Functions of the nodes

Each circuit has n_i primary inputs and n_o outputs. Two additional inputs are reserved for power rails (V_{GND} and V_{DD}). The following encoding scheme is utilized. The primary inputs and node outputs are labeled from 0 to $2 + n_i + n_c \cdot n_r - 1$. The candidate solution is represented by $n_c \cdot n_r$ triplets (x_1, x_2, f) determining node function f, and two positions where node's inputs are connected to. Negative value of x_1 or x_2 is allowed too, but only for one of primary inputs (labeled 2 to $n_i + 1$). When negative value is detected, an implicit inverter of the input is used and inverted value is connected. The last part of the chromosome contains n_o integers specifying the nodes where primary outputs are connected to.

Figure 2 demonstrates the utilized encoding on using XNOR circuit implemented by pass-transistor logic. Implicit inverters are used for each input. In total, the circuit uses 4 implicit transistors to invert each input and 4 explicit transistors for the logic.

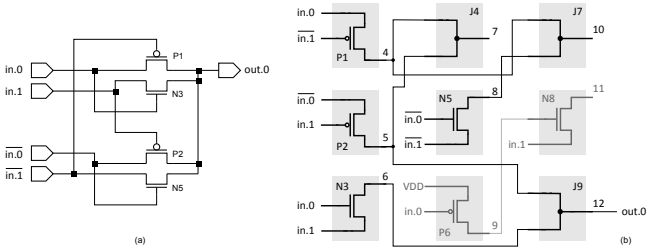

Figure 2: Evolved XNOR circuit and its representation. Chromosome of the solution is *(2,-3,pmos)(-2,3,pmos)(3,2,nmos)(4,5,junction)(-3,-2,nmos)(1,2,pmos)(4,8,junction)(9,3,nmos)(5,6,junction)(12)*.

2.2 Fitness calculation

A candidate circuit with n_i inputs is tested with all possible input vectors, i.e. 2^{n_i} test vectors are applied. The fitness checks the functionality constraints and if they are satisfied then the size (transistors count) or power is optimized. The evaluation is divided into three steps: (i) detection of active nodes, (ii) simulation of each input vector and (iii) power estimation.

2.2.1 Detection of active nodes. Circuit response depends on the so-called active nodes; the others are inactive. The active node is every node either connected directly to output, or acting as transistor with output connected to another active node, or acting as junction with any pin connected to other active node. It is necessary to detect these inactive nodes, because short circuit exception raised by inactive node can be ignored.

At the first, all nodes are marked as inactive. We mark as active nodes only those nodes whose output is connected directly to primary outputs of circuit. Then, in every iteration, we mark as active nodes those nodes that have their output directly connected to some active node input. If the node acts as junction then the connection from the input to the active node's output is allowed too. This step works in linear complexity.

2.2.2 Function verification. The function verification is done by simulating all input vector combinations and comparing the circuit responses with a target truth table. If these results meet the definition, we denote the circuit as valid. Our approach for simulation of circuits is based on event-driven multilevel simulation. We set up all primary inputs including supply voltage. Each node with fully specified inputs recalculates its output value and propagates it to all related nodes. This approach is convenient for speed, because responses are recalculated only for nodes with changed input conditions.

Unlike in standard CGP representation for gate-level digital circuit design, bidirectional data-flow have to be supported. It means that the transistors can pass data from source to drain as well as from drain to source – both electrodes can act as input or output. Junction can cause data-flow in any direction. When a value is propagated from input pin to output pin of previous node in backward direction, there can be more values reduced into one because of *n:1* relation. If every connected wire has the same value (degraded or not) or is in the high impedance state, the strongest value is assigned to wire. In this case, the strongest value is propagated to all other wires which have not the strongest one. If the values are incompatible, short circuit exception is raised and the simulation fails.

Behavior of transistor nodes follows the TSMC 180 nm numerical model. The decision table for calculating drain output from source

Table 1: Function of MOSFET transistor used in simulator. Symbol '1' represents V_{DD} voltage, 'H' represents degraded 1 which is $V_{DD} - V_{tn}$, 'L' is degraded 0 V_{tp}, '0' is logical zero, 'Z' is high impedance state and 'X' undefined.

gate \ source	1	H	L	0	Z	X
1	H	X	L	0	Z	X
H	X	X	L	0	Z	X
L	Z	Z	Z	Z	Z	X
0	Z	Z	Z	Z	Z	X
Z	Z	Z	Z	Z	Z	X
X	X	X	X	X	X	X

nmos

gate \ source	1	H	L	0	Z	X
1	Z	Z	Z	Z	Z	X
H	Z	Z	Z	Z	Z	X
L	1	H	X	X	Z	X
0	1	H	X	L	Z	X
Z	Z	Z	Z	Z	Z	X
X	X	X	X	X	X	X

pmos

and gate electrodes is shown in Table 1. There are two simplifications against precious simulation. High-impedance state is not allowed on the gate, but it is not tested until the end of simulation. When degraded value is located on gate electrode, the transistor is partially opened, but the output value is set to high-impedance.

When a new value of drain is calculated, the current value of drain is checked. If the old one is not compatible with the new one, short circuit is detected. Otherwise the strongest value is propagated to all related nodes. In addition, that state with high impedance source and drain setting can be caused. In this state the transistor works in reverted function (drain as input, source as output) and source is counted similarly as the drain counting was showed.

There is no difference between pins, when the node acts as junction. The node gets three values and if they are compatible the strongest one is propagated to all pins. If one of them is undefined 'X', this value is propagated to all other pins. If values are incompatible short circuit is detected.

2.2.3 Power consumption estimation. One of the most accurate and straightforward methods for the power consumption estimation is to perform a circuit simulation by means of the SPICE simulator. However, it was shown that the simulation results are usually strongly pattern-dependent [4]. Hence large numbers of the input patterns would have to be simulated. This can become computationally very expensive, especially for large circuits. During the evolutionary optimization, it is requested to perform the evaluation quickly.

To avoid the time-consuming exhaustive simulation, we used a probabilistic method for active mode power estimation. The problem of the power consumption estimation can be transformed to the task of computing steady-state transition probabilities. In order to estimate the power consumption, we generalized the approach introduced at gate-level [9] and implemented it for transistor-level circuits [11].

The switching activity is determined according to the signal and transition probabilities obtained from functional verification. Hence, no additional computational overhead is introduced. The signal probability $P_n(x = V)$ at a signal x is defined as the average fraction of clock cycles in which the steady state value of x is equal to the logic value $V \in \Lambda$, where Λ is a set of possible logic values, in our case, six logic values are considered. Let state $S(x) = (g, sd)$ of a transistor node x be defined by the actual logic values present at the signals corresponding to its three terminals, where $g, sd \in \Lambda$.

Given a state $S(x)$, we can calculate a transistor state probability $P_s(S(x) = y)$ for each $y \in (a, b)$ defined as the average fraction of the clock cycles in which a transistor node x remained in the state $S(x)$, where $a, b \in \Lambda$. This simplification can be introduced thanks to the fact that the simulation-based approach inherently takes into account the correlation caused at internal nodes in the circuit due to reconvergence of the input signals or reconvergent fan-out.

Given the transition probabilities, the power consumption can be calculated as:

$$Pwr_{est} = \sum_{\forall x} \sum_{A,B \in \Lambda \times \Lambda} C_{load}(x) Pwr(A, B) P_{tr}^{A \to B}(x),$$

where $C_{load}(x)$ is a constant related to the load capacitance being charged/discharged in the transistor node x and $Pwr(A, B)$ is a constant related to the power consumed by a transistor running at frequency f when its state changed from A to B. Note that Pwr is a technology dependent factor that has to be characterized in advance using a SPICE simulator for n-MOS and p-MOS transistors. The average power consumption measured using SPICE is shown in Figure 3.

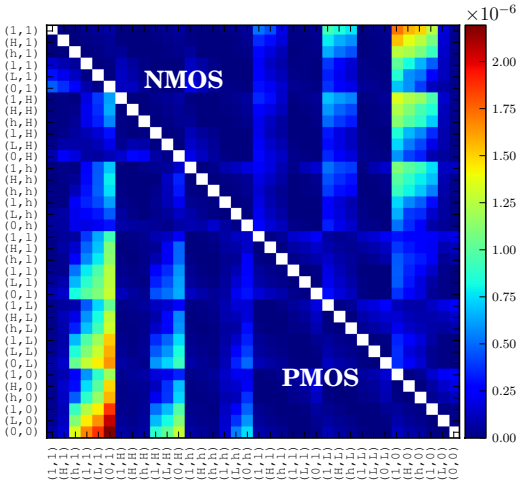

Figure 3: The average power consumption of n-MOS (p-MOS) transistor when its state represented as (g,sd) changed to state (g',sd') at 100MHz and 180nm.

2.3 Search strategy

As search algorithm, the evolution strategy (1+λ) is utilized [6]. The initial population is seeded with a fully-working solution that was obtained from a common gate-level description implementation in which we replaced all gates with theirs CMOS implementation. Every new population consists of the best individual and λ offspring derived from the parent. Fitness (size, power) of each offspring is calculated if functional constraints are not violated. If the obtained fitness is better or equal the best one, the candidate solution is selected as a parent in next population. The offspring are derived from the parent by mutation of a few integer values. As stated in previous section, not all nodes are counted and hence the time of simulation is variable and the time of evolution is not linearly dependent on generations count.

As the discrete simulation uses control instructions primarily and 6-state logic, vectorization using SIMD instructions does not bring any advantages. There are lot of conditional jumps in the resulting code. For that reason we are not able to parallelize the application using AVX units or modern GPU accelerators. Circuit evaluation was already accelerated using special FPTA [14] or FPGA chips [10], but our approach requires running SPICE simulator which is not possible on these chips.

2.4 Single-thread implementation

The proposed evolutionary algorithm for optimization of digital circuits described at the transistor level was implemented using C++

language for standard 64b architecture. The algorithm implements a single-objective Cartesian genetic programming. Please note that all analyses of results were performed in the optimization of one-bit full adder.

We can divide the program into two main parts — evolutionary algorithm and circuit simulator. Evolutionary algorithm uses $(1+\lambda)$ evolutionary strategy. However, the bottleneck of the algorithm is the second part. Our approach is based on combining fast but inaccurate discrete simulation with the state-of-the-art simulator (slow, but accurate SPICE). It helps us to cross the reality gap which is mainly caused by the timing (which means that evolved circuits are correct for a given frequency, but fail for higher frequencies – this is impossible to detect using the proposed fast simulator).

Evolutionary part. In contrast with the standard CGP implementation for the gate-level design, we have to take into account additional constraints when representing candidate circuits. For example, it is not allowed to connect any primary input to the node acting as junction. Otherwise, an incorrect value will be propagated to the primary input if the node's output is changed.

In transistor-level circuit design, there are two additional options how to interconnect circuit components. The circuit primary inputs are usually connected to the gates of transistors. However, in a special type of circuits, the so-called *pass-transistor logic*, it is allowed to connect the inputs to the sources or drains. It causes bigger input load capacitance and in some cases, this phenomenon is undesirable. The second option is to explicitly add inverters to the primary inputs in order to make the problem easier for the evolution. These above-mentioned issues lead to more complex mutation operator.

Simulation part. The discrete simulation is combined with numeric simulation using SPICE. At the beginning, the seed is stored as a valid solution. All candidate circuits are one-by-one evaluated using discrete simulator. The current best circuit is evaluated using SPICE every 800 generation. If the SPICE simulation fails, the run is reverted to the last valid solution and the next population is generated from this one.

3 SHARED MEMORY VERSION

In single-threaded version we used just one core on the CPU. As a first step a multi-threaded implementation of our evolutionary circuit design algorithm was implemented. The usage of shared memory with OpenMP library is one of possible ways how to do it.

3.1 Implementation

We parallelized the application at two levels. The first one evaluates p candidate circuits using p cores in parallel. The second one generates work for the unused cores during SPICE evaluation. Note that only s SPICE simulations can be active as we have only s licenses for SPICE simulator. The thread level parallelism is shown in Figure 4.

The main problem in the population-level parallelism is a variance of the time of fitness calculation. The time is not constant for each candidate solution because some circuits can raise short-circuit exception or can be significantly smaller. It was measured,

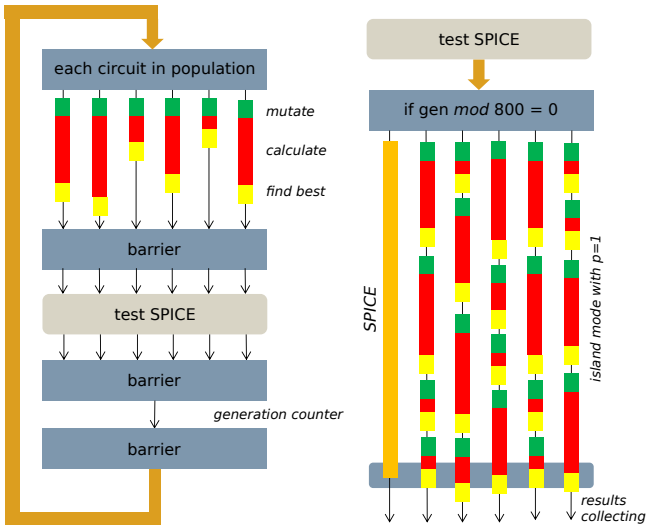

Figure 4: Shared memory implementation. Parallel evaluations of candidate circuits (left); Handling the SPICE simulation (right).

that just about 30% of candidate solutions were valid w.r.t. specification. Because of that 60% of CPU time was unused although the population size was larger than thread count and we used dynamic scheduling.

The SPICE-evaluation parallelisms works as follows. One thread runs the SPICE evaluation. The remaining threads try to incrementally increase the fitness of the candidate solution which was send to SPICE. When the fitness is increased, the circuit is distributed to the other threads. When the SPICE simulation is finished s and the specification is satisfied, the best solution is used as shown in Fig. 4. Otherwise we use previous valid solution and the parallel run is rolled back.

3.2 Results

We performed two tests – with and without SPICE simulation. If SPICE is not used, results are reported for population-level parallelization. The resulting speedup is shown in Table 2.

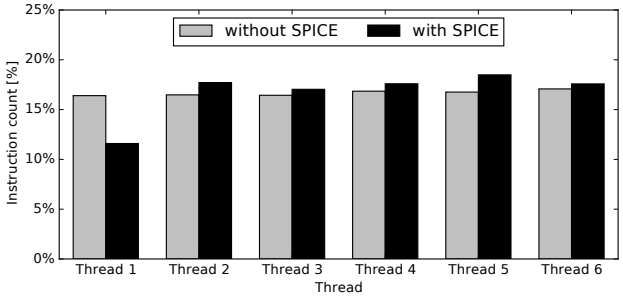

Figure 5: Work distribution for shared-memory version

As we can see, adding more threads does not accelerate the optimization, because a barrier has to be used to synchronize all

Table 2: Speedup of shared memory implementation depending on population size (p) and threads count

(a) without SPICE

p	threads				
	1	2	3	4	6
1	1.00	0.96	0.97	0.94	0.92
6	1.00	1.47	1.52	1.59	1.66
12	1.00	1.47	1.63	1.67	1.68
18	1.00	1.51	1.71	1.73	1.68

(b) with SPICE

p	threads				
	1	2	3	4	6
1	1.00	0.89	0.87	0.87	0.80
6	1.00	1.09	1.15	0.96	1.11
12	1.00	1.14	1.32	1.33	1.25
18	1.00	1.23	1.38	1.40	0.95

threads when candidate circuits are evaluated. We analyzed thread balance using HW counters (Figure 5). The threads are balanced in ±0.3% for 6 threads without SPICE simulation. Note that the implementation employing SPICE simulator is balanced too, except Thread 1 which was used for the numeric simulation (SPICE) with a completely different instruction-mix. Therefore, the poor acceleration is a consequence of the huge variance in time required for circuit evaluation.

On the other hand, parallel processes during SPICE evaluation increase total count of evaluation. This fact helps the algorithm in the searching process. The impact of threads adding is shown in Figure 6.

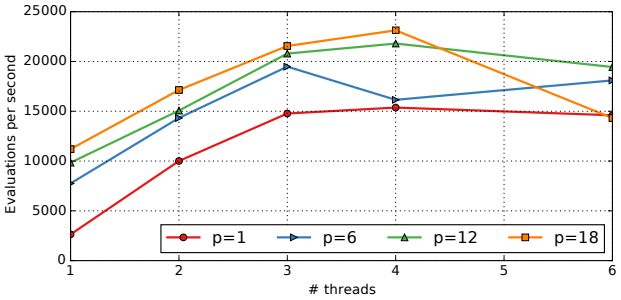

Figure 6: Evaluations per second for different threads count and population size (p)

The results were measured on Intel Xeon processor with 4/8 cores. We can see that 6 threads shows the lowest performance because of hyper-threading. Similarly, performance of 4 threads and $p = 6$ is lower, because 2 threads stop their work when they are waiting for results from the others. Note that speedup in the speed greater than 1 for $p = 1$ w.r.t. 1 thread is caused by changing the evolutionary strategy from $(1+\lambda)$ to $p-1$ parallel $(1+1)$. In general, shared-memory model does not speed up the population-level part, but significantly increases the number of evaluations that can be performed within the same time.

4 ISLAND AND HYBRID VERSIONS

As it was reported in previous section, increasing the number of threads does not bring desired acceleration. In this section, we present a different approach which is called the *island parallelisms*. The main idea is that a set of populations is independently evolved, but the best solutions can migrate among the populations during the evolution. This approach can be implemented by message sending using MPI interface[1]. We also proposed a hybrid version that combines this approach with shared-memory implementation proposed in previous section.

4.1 Implementation

The hybrid implementation is based on the shared-memory implementation. Evolution on one island is implemented using one-thread. The main idea is to exchange the best candidate solution among the populations. To keep the diversity, the solution is accepted only if its fitness (i.e. power consumption calculated using SPICE) is better than the fitness of current best individual in the destination population. When SPICE simulation fails (functional and timing constraints are violated), the best circuit is rolled-back to the previous valid and the circuits obtained by parallel run are refused too.

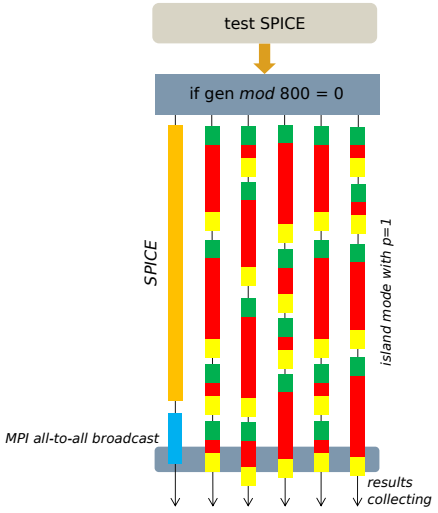

Figure 7: SPICE evaluation with broadcasting of the best solution

The key operation is MPI all-to-all broadcast. We used blocking version of broadcasting, which is waiting for results from all MPI threads. Since sending a small messages may take a long time, we send the fitness and integer netlist (i.e. the best circuit) in one message instead of distributing just the fitness value followed by successive sending the best circuit. The overall scheme is shown in Figure 7.

[1]Message Passing Interface

4.2 Results

This implementation was tested on a computer cluster with following setup. Each node contains 2 CPUs Intel E5, 8 cores each (without hyper-threading). Totally there are up to 16 cores which can use shared-memory model. We used 6 nodes, 144 cores in total. The goal was to optimize the 1-bit full adder. We run evolutionary algorithm for 100 000 generations, i.e. 125 SPICE runs and 125 MPI all-to-all broadcast messages. We performed 3 independent runs for every shared memory group size and population size $p \in \{12, 15\}$.

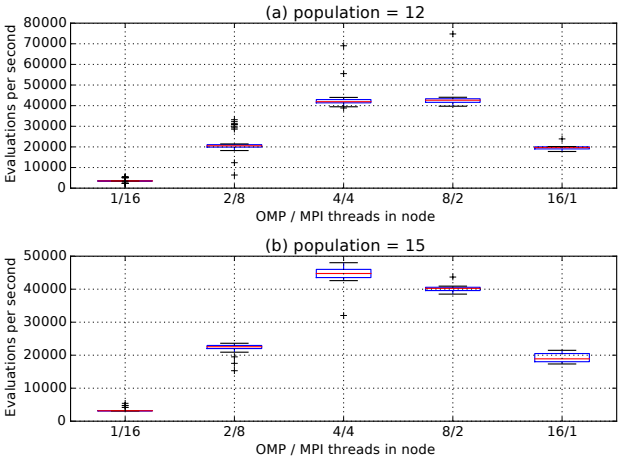

Figure 8: Evaluation step with best solution broadcasting

Figure 8 shows performance (evaluations per seconds) for two population sizes and different shared-memory groups' sizes. The number of threads in this group is denoted as *OMP*. We can see that single OMP thread implementation shows low performance similarly to the pure shared-memory version. If the number of OMP threads is greater than 4, the cache increases the miss rate and with more than 8 threads the memory must be synchronized and the performance falls off. It seems that the ideal proportion is 4/4.

Table 3 compares the results from the evolutionary design conducted with different parameters setting. Note that the ratio of successful updates of circuits between islands is averaged for all runs and the minimal circuit size is found. It is interesting that balanced load of both types of cores on the node leads to more exchanges between nodes. The best exchange rate is in configuration 2/8 where 48 independent islands change the best candidate circuit in 57% of 125 exchanges (6000 messages).

We can see in Table 3 that the most compact solution contains 14 transistors. We compared this solution and a common solution obtained by transferring gate-level netlist to CMOS implementation which requires 48 transistors. The size reduction is up to 70%. In addition the evolved solutions are similar to a 14 transistor another human-created solution [13].

The usage of more MPI threads does not lead to better solutions. The hybrid implementation provides better results and cores utilization than shared memory implementation. One disadvantage is that we need many instances of SPICE simulator which could be problematic if the number of instances is limited.

Table 3: Results of one-bit full adder optimization using hybrid implementation

node	islands	#	succ. updates		# trans.	
omp / mpi		msg	p=15	p=12	p=15	p=12
1 / 16	96	12000	16%	11%	21	20
2 / 8	48	6000	18%	57%	14	14
4 / 4	24	3000	17%	29%	15	14
8 / 2	12	1500	9%	12%	14	15
16 / 1	6	750	10%	15%	18	15

5 GRID IMPLEMENTATION

As stated in previous section, the hybrid implementation provides good results but we need many instances of SPICE simulator. In previous testing, we used open-source simulator ngspice, where the number of instances is unlimited. But sometimes we need to perform better analysis, e.g. power consumption measurement, which is not supported by open-source tools. Hence we need to use some commercial SPICE simulator, but we have to accept higher costs.

5.1 Implementation

If the number of SPICE instances is limited, the approach sharing SPICE instances can be used. Employing the shared-memory model is not practical since it was found as ineffective for runs without SPICE evaluation. The main idea of the client-server algorithm is a connecting of two iteration loops – the first one denoted as *fast improvement* is standard evolutionary strategy $(1+\lambda)$ using discrete simulator. This loop uses random mutation to change the candidate circuit. The second one uses SPICE simulator and the circuits are generated using the fast loop. The advantage of this approach is that the circuits sent to the SPICE have the higher probability to be valid if the discrete simulation works fine. It helps us to deal with uncertainty of discrete simulator without loss of performance. The overall scheme of both parts of this algorithm is shown in Figure 9.

Server part. The goal of server application is receiving of candidate circuits described using integer netlist and performing their SPICE evaluations. As a communication interface the YAMI[2] framework was selected. This framework controls peer-to-peer communication using messages. The main advantage of this framework is the support of a many programming languages and the implementation of all types of communication — blocking and non-blocking (pooling).

The server works as follows. It transforms the integer netlist to SPICE netlist and performs the first testing with all input vector combinations. Note that only V_{DD} and 0 values (logical '0' and '1') are allowed as an input. All 2^{n_i} steps are simulated. If the circuit passes this test, the power analysis starts. The power analysis follows the power estimation method presented in [4]. The results

[2] http://www.inspirel.com/yami4/

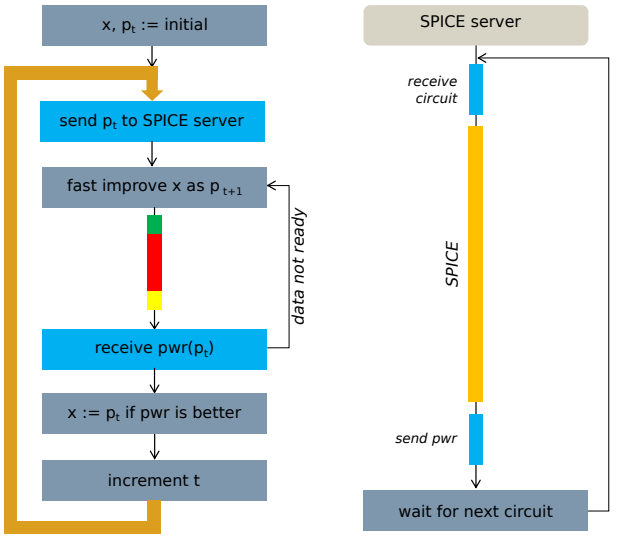

Figure 9: Grid implementation of the client implementing search algorithm and the server evaluating candidate solutions using SPICE.

of both analysis are replied to the sender. Then the next circuit is processed.

Client part. This application runs standard evolutionary strategy ES for CGP. It generates p circuits from the best one and incrementally improves the circuit to get the least power consumption of the circuits that meet functional constraints. The algorithm sends its best solution to SPICE server for validation. During non-blocking waiting for results, ES is running. If the validated solution meets the specification and the real power consumption is better than previous one, this new circuit is selected for the new incremental improvement. The server for evaluation is allocated statically on the startup.

5.2 Results

As one-bit full adder ($n_i = 3, n_o = 2$) was successfully evolved using previous implementation, we evaluated the client-server implementation with a significantly harder task which is optimization of 2 and 4-bit multiplier ($n_i = 4/8, n_o = 4/8$).

The test scenario was as follows. We used a computer cluster where up to 75 threads of client application were running in parallel. The server was running in 15 instances, i.e. each server must serve 5 clients. Servers were distributed to different machines so each server uses one core without any sharing. Totally we run 500 independed client applications for 30 minutes. During these runs 1,368,164 generation with population size $p = 5$ were evolved in average. Each client has 55.32 requests to the server in average and 56% of tested circuits met the functional constraints. We obtained 3800 evaluations per second which corresponds with shared-memory implementation with one thread and $p = 6$. The performance is two-times smaller than the mentioned implementation for one-bit full adder optimization because an addition of one primary input (see Figure 6).

The resulting power distribution for 2 bit multiplier is shown in Figure 10. The initial solution consumes 115.3 μW and consists of 54 transistors. We can see that some circuits were not optimized but some were power-optimized up to 91.7 μW (-20.4 %). The size was 44.9 transistors in average, the most compact solution that we discovered consists of 34 transistors (-37 % savings). To avoid the starvation of GP, we implemented an advanced scheduling version which tries to optimize best solution from the previous finished runs. After 2000 consequent runs running 30 minutes each the power was optimized up to 87.9 μW (-23.7 %).

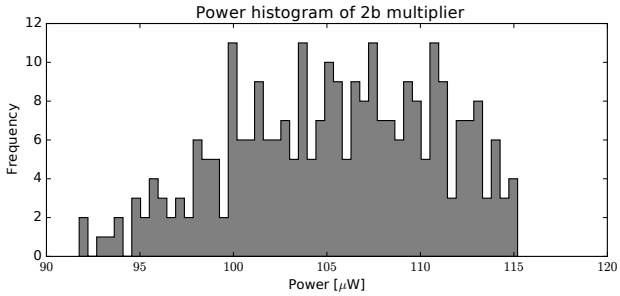

Figure 10: Power distribution of 500 independent 2b multiplier optimization runs for 30 minutes

The goal of the second experiment was to optimize a common 4-bit multiplier. The setup was as follows. Each spice-server serves 3 evolutionary runs, totally 15 servers and 45 clients were running in parallel, 200 clients were running in total. As an initial solution, a compact implementation whose gate-level description consists of 59 two-input gates and exhibits delay of 15 logic gates was chosen. When converted to the transistor-level, the multiplier contains 444 transistors.

Each client evolved the circuit in 15743 generations in the average. The speed of evolution was 8.74 evaluations per second. It is 434 times slower than in the case of 2-bit adder, where 16x less test vectors were tested and 8-13x larger circuits were obtained. In total, 2107 messages were exchanged (i.e. 10.5 per client) and 55 % of them were found as valid, i.e. they met the functional constraints.

Table 4: Power ($\times 10^{-4}$), delay ($\times 10^{-9}$) and power-delay ($\times 10^{-13}$) product for 4-bit multipliers and different technology processes

	Total power			Delay			PDP		
	180	90	45	180	90	45	180	90	45
ORIG	7.95	3.43	1.46	1.69	0.64	0.70	13.43	2.19	1.02
OPT1	7.87	3.38	1.45	**1.57**	**0.58**	**0.70**	**12.36**	**1.96**	**1.01**
impr.	0.9%	1.5%	1.0%	7.1%	9.4%	0.0%	8.0%	11%	1.0%
OPT2	7.65	3.34	**1.41**	1.78	0.70	0.93	13.62	2.34	1.32
impr.	3.7%	2.5%	3.3%	-5.3%	-9.4%	-33%	-1.4%	-6.7%	-28%
OPT3	**7.63**	**3.33**	**1.41**	1.75	0.70	0.93	13.36	2.33	1.31
impr.	4.0%	2.8%	3.8%	-3.6%	-9.4%	-33%	0.6%	-6.3%	-28%

The parameters of the original implementation and the three best discovered alternatives are summarized in Table 4. The total power consumption, worst-case delay and power-delay product (PDP) are included. The best value in each column is emphasized in bold. If we focus only on the results obtained using 180 nm TSMC process, which was used in the fitness function, it can be concluded that the proposed method could improve the total power consumption in all cases. Circuit OPT3 has about 4% lower power consumption compared to the original CMOS implementation. In the case of delay, the first variant labeled as OPT1 exhibits 7% improvement. Overall, OPT1 seems to provide the best results if PDP is considered.

A relative stable reduction in power consumption was achieved if different technology processes are considered. This result indicates that the obtained implementations are relative robust. The delay, however, is very sensitive to the technology process. Only 1% improvement was achieved at 45 nm. It is necessary to note, however, that this result was expected because at lower technology nodes (45nm and below) the leakage current in active mode becomes almost comparable to switching currents. If our goal was to obtain better results for 45 nm technology, it would be necessary to optimize a given circuit using the target (i.e. 45 nm) technology.

6 CONCLUSION

Three new parallel approaches suitable for evolutionary optimization of digital circuits described on transistor-level were introduced in this paper. It was demonstrated that evolutionary algorithm using new discrete simulator can find human competitive solutions. However, while the proposed simulator is fast, it is inaccurate in time domain. This issue was resolved by occasional performing a precise simulation with a commercially available SPICE simulator. Our contributions of the paper are following:

- Shared memory implementation utilizes multi-cores CPU. Although the parallelism for evaluation of candidate circuits during was found ineffective, the evolution during SPICE evaluation significantly improves the performance.
- Island and hybrid implementation combines shared memory model and island evolution with the best circuits exchange.
- Grid implementation tries to overcome limited access to SPICE simulator instances. It divides the problem into two parts — the server side running SPICE and the client running evolutionary optimization. Each server has assigned several clients. This solution is powerful in the term of power optimization due to precise simulation and analysis by SPICE.

Several interesting circuits were discovered during our experiments. We discovered human-competitive implementation of one-bit full adder. We also optimized large circuit such as 2-bit and 4-bit multipliers. The future work can be focused on multi-level evolution combining this approach with gate-level optimization, which can be done effectively using SIMD instructions.

ACKNOWLEGEMENT

This work was supported by the Brno University of Technology project FIT/FSI-J-17-4294.

REFERENCES

[1] Roland Dobai and Lukas Sekanina. 2015. Low-Level Flexible Architecture with Hybrid Reconfiguration for Evolvable Hardware. *ACM Trans. Reconfigurable Technol. Syst.* 8, 3, Article 20 (May 2015), 24 pages.

[2] Yue-Jiao Gong, Wei-Neng Chen, Zhi-Hui Zhan, Jun Zhang, Yun Li, Qingfu Zhang, and Jing-Jing Li. 2015. Distributed evolutionary algorithms and their models: A survey of the state-of-the-art. *Applied Soft Computing* 34 (2015), 286 – 300.

[3] Radek Hrbacek and Vaclav Dvorak. 2014. Bent Function Synthesis by Means of Cartesian Genetic Programming. In *Parallel Problem Solving from Nature - PPSN XIII*. Springer Verlag, 414–423.

[4] Ranjith Kumar, Zhiyu Liu, and Volkan Kursin. 2008. Technique for Accurate Power and Energy Measurement with the Computer-aided Design Tools. *Journal of Circuits, Systems and Computers* 17, 03 (2008), 399–421.

[5] W. B. Langdon and Wolfgang Banzhaf. 2008. *A SIMD Interpreter for Genetic Programming onGPUGraphicsCards*. Springer Berlin Heidelberg, Berlin, Heidelberg, 73–85.

[6] Julian F. Miller (Ed.). 2011. *Cartesian genetic programming*. (22. ed.). Springer, Berlin. 344 pages.

[7] Julian F. Miller, Dominic Job, and Vesselin K. Vassilev. 2000. Principles in the Evolutionary Design of Digital Circuits – Part I. *Genetic Programming and Evolvable Machines* 1, 1 (2000), 8–35.

[8] Julian F. Miller, Dominic Job, and Vesselin K. Vassilev. 2000. Principles in the Evolutionary Design of Digital Circuits – Part II. *Genetic Programming and Evolvable Machines* 1, 3 (2000), 259–288.

[9] J. Monteiro, S. Devadas, A. Ghosh, K. Keutzer, and J. White. 1997. 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 (Jan 1997), 121–127.

[10] Vojtech Mrazek and Zdenek Vasicek. 2014. Acceleration of Transistor-Level Evolution using Xilinx Zynq Platform. In *IEEE Int. Conf. Evolvable Systems, ICES*. Institute of Electrical and Electronics Engineers, 9–16.

[11] Vojtech Mrazek and Zdenek Vasicek. 2015. Automatic Design of Low-Power VLSI Circuits: Accurate and Approximate Multipliers. In *Proc. Int. Conf. Embedded and Ubiquitous Computing*. IEEE, 106–113.

[12] Vojtech Mrazek and Zdenek Vasicek. 2015. Evolutionary Design of Transistor Level Digital Circuits using Discrete Simulation. In *Genetic Programming, 18th European Conference, EuroGP 2015 (LCNS 9025)*. Springer International Publishing, 66–77.

[13] A.M. Shams and M.A. Bayoumi. 2000. A novel high-performance CMOS 1-bit full-adder cell. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on* 47, 5 (May 2000), 478–481.

[14] M.A. Trefzer. 2006. *Evolution of Transistor Circuits*. Ph.D. Dissertation. Ruprecht-Karls-Universitt Heidelberg.

[15] V. Vassilev, D. Job, and J. Miller. 2000. Towards the Automatic Design of More Efficient Digital Circuits. In *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*. IEEE Computer Society, Los Alamitos, CA, USA, 151–160.

[16] Marley Maria Bernard Vellasco, Ricardo Salem Zebulum, and Marco Aurelio Pacheco. 2001. *Evolutionary Electronics: Automatic Design of Electronic Circuits and Systems by Genetic Algorithms* (1st ed.). CRC Press, Inc., Boca Raton, FL, USA.

[17] J.A. Walker, J.A. Hilder, and A.M. Tyrrell. 2008. Evolving Variability-Tolerant CMOS Designs. In *Evolvable Systems: From Biology to Hardware (LNCS)*, Vol. 5216. Springer Berlin Heidelberg, 308–319.

[18] Ludek Zaloudek and Lukas Sekanina. 2008. Transistor-Level Evolution of Digital Circuits Using a Special Circuit Simulator. In *Evolvable Systems: From Biology to Hardware (LNCS)*, Vol. 5216. Springer Verlag, 320–331.