Approximate Computing

Radek Hrbáček
ihrbacek@fit.vutbr.cz

Approximate computing has recently emerged as a promising approach to digital systems design with respect to energy-efficiency or fault-tolerance. 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 energy-efficiency or fault-tolerance improvements can be acchieved by relaxing the need for fully precise operations. With the growing popularity of portable multimedial devices (e.g. smartphones, tablets, etc.), great scope for approximate computation is arising, since human perception is limited and the users are ready to tolerate degraded quality of the multimedial content (e.g. video playback) in exchange for longer battery life. Automatical approximate computing techniques are being developed to speed-up the design process and to find the best trade-off between the resources being shrinked (e.g. energy, time, area) and the inaccuracy of the computation.

The development of automatical design methods infers the need of a mathematical model for digital logic circuits. Inspired by computational complexity theory or circuit complexity theory, *Boolean circuit* model seems to be suitable for this purpose. Since a formal language can be decided by a family of Boolean circuits, common theoretical findings regarding formal languages or automata can be applied.

For the purpose of a formal definition of Boolean circuit, Boolean function has to be defined. A Boolean function is a function of the form $f : \mathrm{D}^k \to \mathrm{D}$, where $\mathrm{D} = \{0, 1\}$ is a Boolean domain and $k \geq 0$ is the arity of the function. The Boolean circuit with n_i inputs and n_o outpus over a basis set B of Boolean functions, representing the logic gates allowable in the circuit model, is defined as a finite directed acyclic graph. Each vertex corresponds to either one of the inputs or a basis function, exactly n_o vertices are labeled as outputs. The basis set B can comprise common logic gates $\{\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}\}$, single NAND gate or other logic gates of arbitrary complexity.

In order to quantify the accuracy loss, we have to introduce proper metrics for approximate computing. The obtained benefit in terms of energy, time, area or other savings has to be measured as well. Each application area has its own specifics and thus the metrics have to be conscientiously chosen with respect to the application specifics to achieve desired results. In the case of general logic circuits, the ratio of incorrect outputs to the total number of outputs (*error rate*) can be used as a metric of accuracy. This can be insufficient for some other applications, e.g. arithmetic blocks, thus a generalized metric *error distance* given by the arithmetical distance between an inexact output and correct output has been proposed.

Since the approximate circuit design can be seen as an (multi-objective) optimization problem, it is possible to exploit evolutionary algorithms (or other biologically inspired techniques) to solve this problem. Cartesian genetic programming (CGP), a variant of genetic programming with a structure very similar to Boolean Circuits, has recently been used to design energy-efficient adders, polymorphic FIR filters or polymorphic image filters. Using multi-objective CGP, multiple constant multipliers (digital circuit which multiplies its single input by N constants) offering Pareto-optimal trade-offs have been designed.

Unfortunatelly, for large-scale circuits (e.g. adders or even worse multipliers) with a high number of inputs, the evaluation can be very time demanding because of a very high number of test cases. In order to speed-up the design process, several techniques can be used including common parallel programming techniques (custom hardware accelerators, computer clusters), coevolutionary principles or formal verification methods (e.g. SAT solvers).

In the upcoming presentation, brief introduction to approximate computing, its formal definition and applications will be given. Great emphasis will be put on formal methods in approximate computing, especially on SAT solvers and their usage for circuit correctness checking (or more precisely for checking if the output value is in given tolerance range).