Approximate Computing and Approximate Circuits (tutorial)

Lukáš Sekanina and Zdeněk Vašíček
sekanina@fit.vutbr.cz; vasicek@fit.vutbr.cz

NASA/ESA AHS 2018, Edinburgh
• Part I: Approximate computing with approximate circuits (Lukáš Sekanina)
  • Motivation for approximate computing
  • Basic principles and techniques
  • Ad hoc circuit approximation
  • Automated circuit approximation
  • Evolutionary circuit approximation
  • Case studies

• Part II: Formal error analysis methods for approximate computing (Zdeněk Vašíček)
  • Error metrics
  • BDD-based error analysis
  • SAT-based error analysis
  • Case studies

• We do not cover
  • approximate memory, cross-layer approximations, bottom-up approximations, approximations in SW ...
Part I:
Approximate computing with approximate circuits
Approximations have always been with us

- Computer engineering
  - FP numbers, computer arithmetic, ...

- Theoretical computer science
  - polynomial time approximation algorithms to find approximate solutions to NP-hard optimization problems

- Stringology, bioinformatics
  - approximate string matching

- Bio-inspired models in AI
  - approximation of functions using artificial neural networks

- Mathematics
  - approximation of functions; numerical mathematics

Why Approximate Computing again?
Why Approximate Computing?

• **Energy efficiency**
  • High performance AND low power computing is requested
    • Big data processing in data centres, supercomputers ...
    • IoT, mobile devices with limited power budget

• **Variability issues**
  • Many “unreliable” components on chips fabricated with modern technologies
  • Standard FT mechanisms are expensive.
  • “Reliable computing” with “unreliable components”.

• **Error resilience**
  • Many applications are error-resilient.
  • We can tolerate these errors!
  • Example: image processing
Approximate computing: Exploiting error resilience

Recognition, Mining, Synthesis Application Suite

- Document search
- Image search
- Digit recognition
- Digit model generation
- Eye detection
- Eye model generation
- Image segmentation
- Census data modeling
- Census data classification
- Health information analysis
- Character recognition
- Online data clustering

Applications have a mix of resilient and sensitive computations

83% of runtime spent in computations that can be approximated


Courtesy of K. Roy
What is Approximate computing?

“Approximate computing exploits the gap between the level of accuracy required by the applications/users and that provided by the computing system, for achieving diverse optimizations.”
[Mittal S., ACM Computing Surveys 2016]

“The requirement of exact numerical or Boolean equivalence between the specification and implementation of a circuit is relaxed in order to achieve improvements in performance or energy efficiency.”
[Venkatesan et al., 2011]

“Computing efficiently by producing results that are good enough or of sufficient quality.”
[Venkataramani et al., DAC 2015]
Approximate computing

- The concept of approximate computing has been developed in different ways and at various levels of the computer stack (circuit, component, memory, processor, compiler, application ...)

- **Software-level approximations**
  - Extensions of general purpose languages (Java, Verilog) to support approximations in data types, operators, ... e.g. EnerJ, Axilog, ExpAx ...
  - Neural network replaces a piece code [Esmaeilzadeh el al., 2013]

- **Specialized processors supporting approximate computing**
  - Improving Efficiency of Extensible Processors by Using Approximate Custom Instructions [Kamal et al., 2014]

- **Circuit approximation**
  - voltage over-scaling, over clocking
  - functional approximations

- **Memory approximation**
  - approximations in memory cells, organization, access, hierarchy ...
Approximate computing in a nutshell (A. Burg, DTIS’16)

**Top-Down**

Objective: Improve Energy Efficiency

---

**Bottom-Up**

Main Idea: Utilize application’s error resiliency to address hardware induced errors

---

**Classical**

Main Idea: Reduce the complexity of an algorithm.

**Techniques**
- Scale down bit-precision
- Prune computations
- Simplify algorithms

**Metrics**
- Quality (SNR, PSNR, ...)
- Energy

---

**New**

**Techniques**
- Allow and tolerate errors
- Limit the impact of errors
- Ensure graceful performance degradation

**Metrics**
- Quality (SNR, PSNR, ...)
- Energy
- Yield
- Reliability (e.g. MTTF)
Sensitivity analysis

• The goal is to identify subsystems suitable for undergoing the approximation.
• Method: Random/guided modification of the original implementation and statistical evaluation of the impact on the quality of result.

In software
• precision of number representation
• data storage strategies
• code simplification
• relaxed synchronization
• unfinished loops
• skipped function calls

In hardware
• bit width reduction
• intentional disconnecting of components
• timing changes
• power supply voltage changes
• fault injection

Chippa et al., ACSSC 2013
Examples of Error metrics

Arithmetic error metrics

- Worst-case error (error magnitude, error significance)
- Relative worst-case error
- The average-case error (average error magnitude, mean error distance)

Generic error metrics

- Error probability (error rate)
- Maximum Hamming distance (bit-flip error)
- Average Hamming distance

Application-specific error metrics

- PSNR
- Distance error
- etc.

\[ e_{\text{wst}}(f, \hat{f}) = \max_{x \in B^n} | \text{int}(f(x)) - \text{int}(\hat{f}(x)) | \]

\[ e_{\text{rel}}(f, \hat{f}) = \max_{x \in B^n} \left| \frac{\text{int}(f(x)) - \text{int}(\hat{f}(x))}{\text{int}(f(x))} \right| \]

\[ e_{\text{avg}}(f, \hat{f}) = \frac{1}{2^n} \sum_{x \in B^n} | \text{int}(f(x)) - \text{int}(\hat{f}(x)) | \]

\[ e_{\text{prob}}(f, \hat{f}) = \frac{1}{2^n} \sum_{x \in B^n} [f(x) \neq \hat{f}(x)] \]

\[ e_{\text{bf}}(f, \hat{f}) = \max_{x \in B^n} \left( \sum_{i=0}^{m-1} f_i(x) \oplus \hat{f}_i(x) \right) \]

\[ e_{\text{hd}}(f, \hat{f}) = \frac{1}{2^n} \sum_{x \in B^n} \sum_{i=0}^{m-1} f_i(x) \oplus \hat{f}_i(x) \]

\( f, \hat{f} \) – original and approximate solution
\( n, m \) – the number of inputs and outputs
\( \text{int} \) – returns a decimal value from \( m \) bits
Approximation techniques (Mittal S., 2016)

- precision scaling
- loop perforation
- load value approximation
- memorization
- task dropping/skipping
- memory access skipping
- data sampling
- using different program (circuit) versions
- using inexact or faulty hardware
- voltage scaling
- refresh rate reducing
- inexact read/write
- reducing branch divergence in GPUs
- lossy compression
- use of neural networks.
Approximate circuits

• Basic approximation techniques
  • Timing induced approximations
  • Functional approximation

• Approximation methodology
  • Ad hoc (circuit-specific)
  • Automated/systematic (circuit independent)
Timing induced approximations

- Power reduction tricks
  - Assume: Accurate circuit $D_1$ at frequency $f_1$
  - $D_1$ is approximated to $D_2$ which can work at higher freq. $f_2$ ($f_2 > f_1$)
  - But, $D_2$ is operated at $f_1$ with lower $Vdd$ $\Rightarrow$ power saving

- Design techniques
  - over-clocking
  - voltage over-scaling

\[ P_{\text{dyn}} = CV_{dd}^2 f \]

\begin{align*}
Vdd = 1.2V \\
\text{delay target}
\end{align*}

\begin{align*}
\text{path delay}
\end{align*}

\begin{align*}
Vdd = 1.2V \\
f \rightarrow 2f
\end{align*}

\begin{align*}
\text{path delay}
\end{align*}

\begin{align*}
Vdd = 0.9V \\
\text{delay target}
\end{align*}

\begin{align*}
\text{path delay}
\end{align*}

\begin{align*}
\text{Timing errors}
\end{align*}

\begin{align*}
\text{Timing errors}
\end{align*}

Courtesy of K. Roy
Functional approximation of digital circuits

Original design:
gate level / RTL / behavioral

Approximate circuit

Quality metrics, constraints, data

Functional approximation

Design methodology

Ad hoc [e.g. Kulkarni et al.: J. Low Power Electronic, 2011]

Design automation methods (= some heuristics used)

SALSA (DAC 2012), SASIMI (DATE 2013), ABACUS (DATE 2014), ASLAN (DATE 2014), AIG-Rewriting (ICCAD 2016) ...

Cartesian Genetic Programming (e.g. ICES 2013, IEEE Tr. on EC 2015, ICCAD 2016, DATE 2017, ICCAD 2017)
Automated functional circuit approximation: Classification

- Where is the approximation conducted?
  - Component (e.g. adder) / module (e.g. DCT) / application (e.g. video compression)

- What is the level of abstraction?
  - transistor, gate, RTL, behavioral, abstract representation (e.g. SoP, BDD, AIG ...)

- How is the circuit approximated?
  - truncation
  - pruning
  - component replacement (using a library of approximate components)
  - re-synthesis
  - others

- How are the candidate approximate circuits evaluated?
  - quality (at different levels of the application)
    - simulation/probabilistic/formal-based methods
  - electrical parameters
    - power, delay, area, ...

- How is the approximation method evaluated?
  - The approximation methods are often heuristics! A proper statistical evaluation is requested.
How to determine the error?

Error “estimation”
- (Functional) circuit simulation
- Probabilistic models, e.g. Li at al., DAC 2015
  - applicable in very specific cases only

Exact error calculation
- Exhaustive simulation – small problem instances only
- Analysis of Binary Decision Diagrams (ROBDDs)
  - Worst-case error (M. Soeken et al., ASP-DAC’16)
  - Average error (Vasicek et al., DATE’17)
  - Average Hamming distance (Vasicek, Sekanina, GENP’16)
  - Not scalable for some circuits such as multipliers
- Transforming to SAT problem
  - Worst case error
    - Venkatesan et al. (ICCAD’11), Petkovska et al. (ICCAD’16), Ceska et. al. (ICCAD’17) ...
  - Not suitable if counting the number of solutions is requested.
On a fair comparison of automated approx. methods

- **Common practice**: The original circuit and approximate circuits created using a given method are compared -> *not sufficient!*
- A comparisons with other approximation methods is needed!
- Important assumptions for a fair comparison:
  - the original circuits are the same
  - the error is calculated using the same method (simulation vs. exact)
  - electrical parameters are calculated using the same tool and for the same technology library
  - the time/resources for the approximation methods under investigation are the same
  - the same statistically relevant values are reported (best, median, mean etc.)
Ad hoc approximation approaches

- adders
- multipliers

Approximate adders (ad hoc)

• Speculative Adders
  • For a 128 bit adder, the probability that the carry propagation chain is longer than 12 and 18 are 1% and 0.01%, respectively.
  • Therefore, $k$ bits are used to speculate the carry for each sum bit $(k < n)$.

• Segmented Adders
  • An $n$-bit adder is divided into a number of smaller $k$-bit sub-adders.
  • The carry may be generated by using different methods.

• Carry-Select Adders
  • Multiple sub-circuits are used to compute the sum for different carry values, and the result is selected by the carry of a sub-circuit.

• Approximate Full Adders
## Approximate adders (ad hoc)

<table>
<thead>
<tr>
<th>Adder Type</th>
<th>Adder Name</th>
<th>Delay</th>
<th>Circuit Area</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Conventional Adders</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RCA</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td></td>
</tr>
<tr>
<td>CLA</td>
<td>$O(\log(n))$</td>
<td>$O(n\log(n))$</td>
<td></td>
</tr>
<tr>
<td><strong>Speculative Adders</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACA [6]</td>
<td>$O(\log(k))$</td>
<td>$O((n - k)k\log(k))$</td>
<td></td>
</tr>
<tr>
<td>ESA [12]</td>
<td>$O(\log(k))$</td>
<td>$O(n\log(k))$</td>
<td></td>
</tr>
<tr>
<td><strong>Segmented Adders</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ETAII [7]</td>
<td>$O(\log(k))$</td>
<td>$O(n\log(k))$</td>
<td></td>
</tr>
<tr>
<td>ACAAA [13]</td>
<td>$O(\log(k))$</td>
<td>$O((n - k)\log(k))$</td>
<td></td>
</tr>
<tr>
<td><strong>Approximate Adders</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCSA [8]</td>
<td>$t_{adder} + t_{mux}$</td>
<td>$A_{adder} + A_{mux}$</td>
<td></td>
</tr>
<tr>
<td>CSA [14]</td>
<td>$O(\log(k))$</td>
<td>$A_{adder} + A_{carry}$</td>
<td></td>
</tr>
<tr>
<td>CSPA [15]</td>
<td>$t_{adder} + t_{mux}$</td>
<td>$A_{adder} + A_{mux} + A_{carry}$</td>
<td></td>
</tr>
<tr>
<td>CCA [16]</td>
<td>$t_{adder} + t_{mux}$</td>
<td>$A_{adder} + A_{mux}$</td>
<td></td>
</tr>
<tr>
<td>GCSA [11]</td>
<td>$O(\log(k))$</td>
<td>$O(n\log(k))$</td>
<td></td>
</tr>
<tr>
<td><strong>Approximate Full Adders</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOA [10]</td>
<td>$O(\log(n - l))$</td>
<td>$A_{loa} + (l \times A_{OR})$</td>
<td></td>
</tr>
<tr>
<td><strong>Truncated Adders</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TruA</td>
<td>$O(\log(n - l))$</td>
<td>$O((n - l)\log(n - l))$</td>
<td></td>
</tr>
</tbody>
</table>

$t_{adder}: O(\log(k)) \quad A_{adder}: O(n\log(k)) \quad A_{loa}: O((n - l)\log(n - l)) \quad A_{carry}: \text{circuit area of the carry prediction circuit}$

Han J. ESWEEK tutorial, 2017
Approximate full adder

Conventional (mirror) adder

Approximate adder (Approx. I)

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Approx. I</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Benefits: Fewer transistors, lower dynamic & leakage power, shorter critical path, opportunity for down-sizing

Evaluation (JPEG compression)

60% power savings and 37% area savings with 5.7 dB loss in output quality (PSNR)

Approximate 16 bit adders

16-bit unsigned adders

**MRED** (Mean Relative Error Distance) calculated with Monte Carlo simulation (100 M vectors)

**ER** (Error Rate)

**PDP** – Power Delay Product

STM CMOS 28 nm technology, 1V

Han J. ESWEEK tutorial, 2017
**Ad hoc approximation of multipliers: 2-bit multiplier**

- Correct results for **15 out of 16** input combinations (almost 50% area reduction, lower delay).
- Used as a building block for larger multipliers and then in image processing applications.

**Error probability**

<table>
<thead>
<tr>
<th>Bit-Width</th>
<th>Error-Prob</th>
<th>Mean-Error</th>
<th>Max-Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.0625</td>
<td>1.39%</td>
<td>22.22%</td>
</tr>
<tr>
<td>4</td>
<td>0.19</td>
<td>2.60%</td>
<td>22.22%</td>
</tr>
<tr>
<td>8</td>
<td>0.46</td>
<td>3.25%</td>
<td>22.22%</td>
</tr>
<tr>
<td>12</td>
<td>0.675</td>
<td>3.31%</td>
<td>22.22%</td>
</tr>
<tr>
<td>16</td>
<td>0.81</td>
<td>3.32%</td>
<td>22.22%</td>
</tr>
</tbody>
</table>

**Dynamic power reduction for various frequencies**

<table>
<thead>
<tr>
<th>Bit</th>
<th>F (%)</th>
<th>1.25F (%)</th>
<th>1.5F (%)</th>
<th>1.75F (%)</th>
<th>2F (%)</th>
<th>Avg. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>44.9</td>
<td>42.1</td>
<td>42.1</td>
<td>48.9</td>
<td>48.9</td>
<td>45.4</td>
</tr>
<tr>
<td>4</td>
<td>13.7</td>
<td>31.6</td>
<td>44.8</td>
<td>44.7</td>
<td>46.5</td>
<td>36.3</td>
</tr>
<tr>
<td>8</td>
<td>33.1</td>
<td>40.4</td>
<td>26.3</td>
<td>48.8</td>
<td>58.9</td>
<td>41.5</td>
</tr>
<tr>
<td>16</td>
<td>25.6</td>
<td>29.6</td>
<td>32.4</td>
<td>33.8</td>
<td>37.4</td>
<td>31.8</td>
</tr>
</tbody>
</table>

## Approximate multipliers: Classification

<table>
<thead>
<tr>
<th>Classification</th>
<th>Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approximation in Generating partial products</td>
<td>Under-Designed Multiplier (UDM)</td>
</tr>
<tr>
<td>Approximation in the partial products</td>
<td>Broken Array Multiplier (BAM)</td>
</tr>
<tr>
<td></td>
<td>Error Tolerant Multiplier (ETM)</td>
</tr>
<tr>
<td></td>
<td>Approximate Wallace Tree Multiplier (AWTM)</td>
</tr>
<tr>
<td></td>
<td>Truncated Wallace Multiplier (TruMW)</td>
</tr>
<tr>
<td></td>
<td>Truncated Array Multiplier (TruMA)</td>
</tr>
<tr>
<td>Using approximate counters or compressors</td>
<td>Inaccurate Compressor based Multiplier (ICM)</td>
</tr>
<tr>
<td></td>
<td>Approximate Compressor based Multiplier (ACM)</td>
</tr>
<tr>
<td></td>
<td>Approximate Multiplier 1/2 (AM1/AM2)</td>
</tr>
<tr>
<td></td>
<td>Truncated AM1/AM2 (TAM1/TAM2)</td>
</tr>
<tr>
<td>Approximate Booth multipliers</td>
<td>Fixed-width Booth multipliers</td>
</tr>
</tbody>
</table>

Han J. ES WEEK tutorial, 2017
Approximate multipliers: Evaluation

16-bit unsigned multipliers

MRED (Mean Relative Error Distance) calculated with Monte Carlo simulation (100 M vectors)

STM CMOS 28 nm technology, 1V

Han J. ESWEEK tutorial, 2017
Automated approximation methods
### Automated functional circuit approximation: Classification

<table>
<thead>
<tr>
<th>First Auth., Conf/Journal, Tool</th>
<th>Method, description</th>
<th>Error comp.</th>
<th>Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shin, DATE10</td>
<td>Elimination in SoP</td>
<td>Exhaustive sim.</td>
<td>&lt;16 inputs: rd73, sym10, rd73, clip, sao2, 5xp1, t481</td>
</tr>
<tr>
<td>Shin, DATE11</td>
<td>Greedy, fault injection</td>
<td>Simulation</td>
<td>c880, c1908, c3540, c5315, c7552</td>
</tr>
<tr>
<td>Venkataramani, DAC12, SALSA</td>
<td>Don’t care simplification</td>
<td>SAT</td>
<td>32-bit+, 8-bit *, 8-bit MAC, SAD, BUT, FIR, IIR, DCT</td>
</tr>
<tr>
<td>Venkataramani, DATE13, SASIMI</td>
<td>Similar signal detection</td>
<td>Simulation</td>
<td>ISCAS85, 32-bit +, 8-bit *, MAC, SAD, …</td>
</tr>
<tr>
<td>Ranjan, DATE14, ASLAN</td>
<td>Sequential/heuristics</td>
<td>SAT</td>
<td>FIR, IIR, MAC, DCT, Sobel, SAD, BUT …</td>
</tr>
<tr>
<td>Nepal, DATE14, ABACUS</td>
<td>Greedy over AST</td>
<td>Simulation</td>
<td>FIR, FFT, perceptron, block matcher, …</td>
</tr>
<tr>
<td>Venkataraman, DATE15</td>
<td>Probabilistic pruning</td>
<td>Simulation</td>
<td>Filters, QRS in ECG</td>
</tr>
<tr>
<td>Li, DAC15</td>
<td>Replacement in HLS</td>
<td>Probabilistic</td>
<td>MediaBench, IIR, FIR, …</td>
</tr>
<tr>
<td>Soeken, ASPDAC16, ABM</td>
<td>Heuristics over BDD</td>
<td>BDD</td>
<td>6 ISCAS-85</td>
</tr>
<tr>
<td>Chandrasekharan, ICCAD16</td>
<td>Greedy, rewriting, AIG</td>
<td>BDD, SAT</td>
<td>LGSynth91, 8/16-bit +, 8 bit *, MAC, parity</td>
</tr>
<tr>
<td>Jain, DATE16</td>
<td>Logic isolation</td>
<td>Probabilistic</td>
<td>32-bit +, 12-bit *, 8-bit DCT, FFT, FIR, …</td>
</tr>
<tr>
<td>Lofti, DATE16, GRATER</td>
<td>Truncation, OpenCL</td>
<td>Simulation</td>
<td>Sobel, DCT, recurs. Gaussian, n-body, convolution</td>
</tr>
<tr>
<td>Sekanina, SSCI-ICES13</td>
<td>CGP</td>
<td>Exhaustive sim.</td>
<td>4 ISCAS85 circuits, adders</td>
</tr>
<tr>
<td>Vašíček, IEEE Tr. on EC, 2015</td>
<td>CGP</td>
<td>Simulation</td>
<td>Multipliers, 9/25-input median</td>
</tr>
<tr>
<td>Vašíček, GPEM, 2016</td>
<td>CGP</td>
<td>BDD</td>
<td>Selected circuits from LGSynth, ITC and ISCAS</td>
</tr>
<tr>
<td>Češka, ICCAD17</td>
<td>CGP</td>
<td>SAT</td>
<td>32-bit *, 128-bit +</td>
</tr>
</tbody>
</table>
Finding minterm complements to reduce # literals

- The objective is to obtain designs that have a minimum number of literals for a given error rate threshold.
- Method: Identify minterm complements that produce an approximate circuit version that has the smallest number of literals for a given error rate threshold.
- Exhaustive search for simple functions, a heuristics approach for more complex functions.

Original solution:
$$\overline{x_1}x_2x_4 + x_2x_3x_4 + x_1\overline{x_2}\overline{x_3}x_4$$

Approximation 1: $$x_2x_4 + x_1\overline{x_3}x_4$$

Approximation 2: $$\overline{x_1}x_2x_4 + x_2x_3x_4$$

Shin and Gupta: Approximate logic synthesis for error tolerant applications. DATE 2010
**Key Idea:** Identify signal pairs (TS and SS) that are similar in functionality, i.e., produce the same value for most of the inputs among signal pairs.

- **Substitute** one in place of the other
  - Circuit becomes approximate
- **Simplify** the circuit: Logic Deletion & Downsizing

The signal probability calculation engine in Synopsys Power Compiler was used to obtain difference probabilities.

S. Venkataramani, K. Roy, and A. Raghunathan: Substitute-and simplify: a unified design paradigm for approximate and quality configurable circuits, DATE’13, pp. 1367–1372
Approximation-aware Rewriting of AIGs

- **Principle**: allow AIG rewriting to change the functionality of the circuit without violating a predefined error bound.
- Rewriting (at the level of cuts on selected paths) takes a greedy approach.
- Worst-case error, bit-flip error and error rate determined exactly (formally).
- Evaluated: 8/16-bit adders, LGSYnth91, 8-bit multipliers, 32-bit parity, ...

Heuristics:
- replace the cut by constant 0

2-bit adder

Chandrasekharan, Soeken, Grosse, Drechsler. Approximation-aware Rewriting of AIGs for Error Tolerant Applications ICCAD 2016
ABACUS: Approximations at Behavioral RT-level

- Original file: Verilog
- Abstract Syntax Tree (AST) transformations (mutations)
  - Data type simplification
  - Operation transformations (e.g. + -> or)
  - Arithmetic expression transformation
  - Variable to Constant transformations
  - Loop transformations
- Search algorithm: Greedy / NSGA-II
- Fitness is obtained by circuit simulation and combines the error & power

ABACUS: Results

Benchmark problems:

<table>
<thead>
<tr>
<th>Design</th>
<th>Class of Application</th>
<th>#Lines</th>
<th>Area (um$^2$)</th>
<th>Power (mW)</th>
<th>Quality Measure</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>perceptron</td>
<td>Machine Learning</td>
<td>188</td>
<td>37775.16</td>
<td>2.74</td>
<td>classification error</td>
<td>82.9%</td>
</tr>
<tr>
<td>FIR filter</td>
<td>Signal Processing</td>
<td>265</td>
<td>40390.20</td>
<td>6.89</td>
<td>MSE</td>
<td>99.45%</td>
</tr>
<tr>
<td>FFT</td>
<td>Signal Processing</td>
<td>255</td>
<td>18480.96</td>
<td>2.07</td>
<td>MSE</td>
<td>100%</td>
</tr>
<tr>
<td>block matching</td>
<td>Computer Vision</td>
<td>1277</td>
<td>80272.44</td>
<td>30.42</td>
<td>PSNR</td>
<td>30.66 dB</td>
</tr>
</tbody>
</table>

Results of evolutionary approximation:

Cartesian Genetic Programming (CGP) [Miller, 1999]

- $n_i$ primary inputs
- $n_o$ primary outputs
- $n_c$ columns
- $n_r$ rows

- $n_a$ inputs of each node
- $\Gamma$ function set
- L-back parameter

Nodes in the same column are not allowed to be connected to each other.
No feedback!
CGP: Representation for logic networks

Genotype (netlist):
- \( n_a + 1 \) integers per node; \( n_o \) integers for outputs;
- Constant size: \( n_c n_r (n_a + 1) + n_o \) integers

Phenotype (directed acyclic graph \( \Rightarrow \) circuit):
- Variable size; unused nodes are ignored.

• CGP parameters
  - \( n_r = 3 \) (#rows)
  - \( n_c = 3 \) (#columns)
  - \( n_i = 3 \) (#inputs)
  - \( n_o = 2 \) (#outputs)
  - \( n_a = 2 \) (max. arity)
  - \( L = 3 \) (level-back parameter)
  - \( \Gamma = \{ \text{NAND}^{(0)}, \text{NOR}^{(1)}, \text{XOR}^{(2)}, \text{AND}^{(3)}, \text{OR}^{(4)}, \text{NOT}^{(5)} \} \)
CGP: Fitness function for circuit design

Typical fitness function (circuit functionality):

\[ f = \text{Max} - \sum_{i=1}^{K} \text{HD}(y_i, w_i) \]

The number of test vectors

Hamming distance (between circuit response and desired response)

\[ \text{Max} = \#\text{outputs} \times 2^{\#\text{inputs}} \text{ (in our case } 2 \times 2^8 = 16) \]

\[ K = 2^{\#\text{inputs}} \text{ for combinational circuits. Not scalable!!!} \]

Specification (1-bit adder), target table:

<table>
<thead>
<tr>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\(\Rightarrow\) fitness = 16

<table>
<thead>
<tr>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\(\Rightarrow\) fitness = 10

Additional objectives:
- area (the number of gates)
- delay
- power consumption etc.
- **Mutation**: Randomly select $h$ integers and replace them by randomly generated (but legal) values.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
| 1 | 1 | 1 | 1 | 1 | => fitness = 10

mutation

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
| 1 | 1 | 1 | 1 | 1 | => fitness = 16
(for full adder)
Algorithm 1: CGP

Input: CGP parameters, fitness function
Output: The highest scored individual $p$ and its fitness

1. $P \leftarrow$ randomly generate population; // or use conventional designs
2. EvaluatePopulation($P$); $p \leftarrow$ highest-scored-individual($P$);
3. while (terminating condition not satisfied) do
4.     $\alpha \leftarrow$ highest-scored-individual($P$);
5.     if fitness($\alpha$) $\geq$ fitness($p$) then
6.         $p \leftarrow \alpha$;
7.     $P \leftarrow$ create $\lambda$ offspring of $p$ using mutation;
8.   EvaluatePopulation($P$);
9. return $p$, fitness($p$);
CGP for circuit (functional) approximation

- **Error-oriented** (single-objective) method
  - CGP gradually degrades a fully functional circuit until a circuit with a **required error** is obtained. Then, the area (and so power consumption) is minimized for this error.

- **Resources-oriented** (single-objective) method
  - CGP is used to minimize the error, but only limited resources (components) are provided, insufficient for constructing a fully functional circuit.

- **Multi-objective optimization**
  - All target parameters are optimized together.
Evolved library of 8-bit approx. adders and multipliers

- Comprehensive library of approximate arithmetic circuits
  - 430 non-dominated adders (evolved from 13 accurate adders)
  - 471 non-dominated multipliers (evolved from 6 accurate multipliers)
- Method: Multi-objective CGP with NSGA-II

**Fig. 3** Pareto fronts of the evolved 8-bit approximate adders and multipliers

V. Mrazek, R. Hrbacek, Z. Vasicek, L. Sekanina: EvoApprox8b, DATE 2017
Evolved library of 8-bit approx. multipliers

Fig. 4 Parameters of 8-bit multipliers synthesized using 45 nm technology with $V_{cc} = 1V$.
Evolved library of 8-bit approx. adders and multipliers

Approximate adders (430), exact adders (43)

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Est. area</th>
<th>Est. delay</th>
<th>Est. power</th>
<th>Nodes</th>
<th>HD</th>
<th>MAE</th>
<th>MSE</th>
<th>MRE</th>
<th>WCE</th>
<th>WCRE</th>
<th>EP</th>
<th>OPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>add8_000</td>
<td>820 μm²</td>
<td>1.314 ns</td>
<td>194.31 μW</td>
<td>10</td>
<td>138496</td>
<td>1.71875</td>
<td>6.00000</td>
<td>0.88%</td>
<td>7</td>
<td>100 %</td>
<td>71.875 %</td>
<td>Verilog, Matlab</td>
</tr>
<tr>
<td>add8_001</td>
<td>2040 μm²</td>
<td>0.718 ns</td>
<td>681.20 μW</td>
<td>42</td>
<td>0</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00%</td>
<td>0</td>
<td>0 %</td>
<td>0.000 %</td>
<td>Verilog, Matlab</td>
</tr>
<tr>
<td>add8_002</td>
<td>836 μm²</td>
<td>1.282 ns</td>
<td>194.75 μW</td>
<td>13</td>
<td>140448</td>
<td>1.69531</td>
<td>5.85038</td>
<td>0.88%</td>
<td>7</td>
<td>100 %</td>
<td>71.484 %</td>
<td>Verilog, Matlab</td>
</tr>
<tr>
<td>add8_003</td>
<td>912 μm²</td>
<td>0.379 ns</td>
<td>266.66 μW</td>
<td>20</td>
<td>192640</td>
<td>9.64844</td>
<td>138.25000</td>
<td>5.21%</td>
<td>24</td>
<td>100 %</td>
<td>96.875 %</td>
<td>Verilog, Matlab</td>
</tr>
<tr>
<td>add8_004</td>
<td>708 μm²</td>
<td>1.213 ns</td>
<td>205.54 μW</td>
<td>9</td>
<td>134628</td>
<td>1.37500</td>
<td>3.25000</td>
<td>0.75%</td>
<td>5</td>
<td>200 %</td>
<td>76.562 %</td>
<td>Verilog, Matlab</td>
</tr>
</tbody>
</table>

Approximate multipliers (471), exact multipliers (28)

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Est. area</th>
<th>Est. delay</th>
<th>Est. power</th>
<th>Nodes</th>
<th>HD</th>
<th>MAE</th>
<th>MSE</th>
<th>MRE</th>
<th>WCE</th>
<th>WCRE</th>
<th>EP</th>
<th>OPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>mu8_000</td>
<td>9224 μm²</td>
<td>3.015 ns</td>
<td>4933.22 μW</td>
<td>137</td>
<td>176134</td>
<td>98.52710</td>
<td>27520.00000</td>
<td>1.99%</td>
<td>820</td>
<td>560 %</td>
<td>86.490 %</td>
<td>Verilog, Matlab</td>
</tr>
<tr>
<td>mu8_001</td>
<td>5200 μm²</td>
<td>3.666 ns</td>
<td>2524.84 μW</td>
<td>91</td>
<td>310762</td>
<td>239.95550</td>
<td>108908.84375</td>
<td>5.36%</td>
<td>1671</td>
<td>100 %</td>
<td>98.169 %</td>
<td>Verilog, Matlab</td>
</tr>
<tr>
<td>mu8_002</td>
<td>6715 μm²</td>
<td>2.086 ns</td>
<td>2789.47 μW</td>
<td>132</td>
<td>338906</td>
<td>329.88147</td>
<td>207883.35278</td>
<td>6.70%</td>
<td>2193</td>
<td>700 %</td>
<td>98.482 %</td>
<td>Verilog, Matlab</td>
</tr>
<tr>
<td>mu8_003</td>
<td>4172 μm²</td>
<td>1.983 ns</td>
<td>1816.06 μW</td>
<td>79</td>
<td>376002</td>
<td>624.46875</td>
<td>679898.57422</td>
<td>10.00%</td>
<td>2911</td>
<td>700 %</td>
<td>98.984 %</td>
<td>Verilog, Matlab</td>
</tr>
<tr>
<td>mu8_004</td>
<td>5034 μm²</td>
<td>1.944 ns</td>
<td>1893.73 μW</td>
<td>104</td>
<td>382402</td>
<td>639.22653</td>
<td>709554.15625</td>
<td>9.76%</td>
<td>3143</td>
<td>253 %</td>
<td>99.071 %</td>
<td>Verilog, Matlab</td>
</tr>
</tbody>
</table>

Synthesis results for 45 nm and 180 nm technology (Synopsys Design Compiler), 7 error metrics

http://www.fit.vutbr.cz/research/groups/ehw/approxlib/
Approximate neural networks on a chip

**Approximations can be introduced in:**

- ANN structure – pruning
- Data representation – compression.
- Memory – approximate cells and Load/Store
- Datapath
  - Reducing data bit-width
  - Multiplication in neurons and convolution layers
    - approx. 30-50% of total power
- Activation function
- Sum function

Approximate multipliers for Deep NNs

- Floating point (FP) used for training, fixed point (FX) for inference.
- FX: 8 – 16 bits are usually sufficient (e.g., 8-bit multipliers on TPU)
- Various approximate multipliers have been employed
- **Multiplier-less multiplication** (Alphabet Set Multiplier -
  - \(01100100_2 \times X = (3X \times 2^1) \times 2^4 + (1X \times 2^2) \times 2^0\)
  - Only a subset of weights is used

Case study: Evolution of approximate multipliers for CNNs

MNIST dataset classification: 32x32 – 100 – 10 MLP network (classification accuracy 94.16% with accurate implementation). We introduced an approximate multiplier by adding a jitter function $\Delta(a, b)$, resulting in a 5.2% error for multiplication.

**Scenario A:**
- Multiplication
  $$m(a, b) = a \cdot b + \Delta(a, b)$$
- Classification accuracy: 10.77%

**Scenario B:**
- 80% of multiplications are by 0
- Multiplication
  $$m'(a, b) = \begin{cases} 
    0 & \text{if } a = 0 \vee b = 0 \\
    a \cdot b + \Delta(a, b) & \text{otherwise}
  \end{cases}$$
- Classification accuracy: 94.20%
Accurate multiplier – initial circuit (6)
- CSAM RCA, CSAM RCA, RCAM, WTM CLA, WTM CSA, WTM RCA

Target errors: $\varepsilon \in \{0.5\%, 1\%, 2\%, 5\%, 10\%, 15\%, 20\%\}$

CGP parameters
- $n_i \in \{14, 22\}; \ n_o \in \{14, 22\}; \ n_r = 1; \ 250 < n_c < 780$
- Function set: \{NOT, AND, NAND, OR, NOR, XOR, XNOR\}
- Error constraints:
  1. $\forall a, b: \left|m(a, b) - a \times b\right| \leq \varepsilon \cdot 2^{n_o}$
  2. $\forall a: m(a, 0) = m(0, a) = 0$
- Fitness function:
  \[
  C(m) = \begin{cases} 
  \text{GatesCount}(m) & \text{if constraints (1) and (2) met}, \\
  -\infty & \text{otherwise}
  \end{cases}
  \]
Approximate multipliers (with $0 \times x = 0$) evolved with CGP

Results of synthesis of sign-extended multipliers with Synopsys DC

- 45 nm technology
- Timing:
  - 8-bit multipliers: 2.5 GHz
  - 12-bit multipliers: 2 GHz
- Accurate multiplier was implemented in Verilog using standard $\ast$ arithmetic operator

Evaluation setup – MNIST dataset

- Handwritten number dataset
- Fully connected MLP network
- 28x28 inputs, 300 hidden neurons, 10 outputs
- 60k training images
- 10k testing images
- More than 238k multiplications for approximation
- Initial classification accuracy:
  - 8b: 97.67%
  - 12b: 97.70%
Evaluation setup – SVHN dataset

- Real-world data
- Convolutional LeNet NN
- 278,104 multiplications in 6 layers
- 73k training images
- 26k testing images
- Approximation introduced in L1, L3, L5 and L6 layers
- Initial classification accuracy:
  - 8b: 86.85%
  - 12b: 86.90%
Energy-efficient implementation of ANNs: Summary

Classification Accuracy and power reduction (in multiplication)

Approximation error $\varepsilon$ of multipliers

Power (8 bit) (12 bit)
-20% -30% -57% -77% -82% -91% -91% -36% -25% -9%
-50% -43% -66% -70% -85% -86% -87% -60% -20% -1%

Classification accuracy of NN

0% 0.50% 1% 2% 5% 10% 15% 20% {1} {1,3} {1,3,5,7}

MNIST $w=8$
MNIST $w=12$
SVHN $w=8$
SVHN $w=12$

Multiplierless multiplication by Sarwar et al. DATE’2016

Non-linear image filters

corrupted image
(10% pixels, impulse noise)

filtered image
(9-input median filter)
Non-linear image filters: Approximation strategies

- **Approximation of the comparator element**

- **Approximation of the network (pruning)**
  - CGP used to find a network of $N$ comparators minimizing the error w.r.t. the original median (consisting of $K$ comparators), but resources are limited, i.e. $N < K$.

- **Evolutionary image filter design from scratch**
  - CGP used to evolve an image filter showing a minimal error and cost. Filters are composed of elementary 2-input functions (min, max, +, logic functions over 8 bits).
Approximate median using CGP

- Median network (consisting of up to $N$ operations) is represented by means of a one-dimensional array of $N$ nodes.
- Each node can act as: identity (0), minimum (1), maximum (2) over 8 bits
- Each candidate solution is encoded using $3N + 1$ integers.
- Fitness function (single objective)
  \[ error = \sum_{i \in S} |O_{\text{candidate}}(i) - O_{\text{reference}}(i)| \]
- Example for a 3-input median:

Chromosome: 0, 2, 3; 3, 2, 0; 0, 2, 2; 5, 3, 1; 6, 1, 2; 7, 0, 0; 6, 8, 2; 8
Approximate median using CGP

Experimental setup
- (1+4) pop. size, no crossover, 5% of the chromosome mutated

<table>
<thead>
<tr>
<th></th>
<th>Median-9</th>
<th>Median-25</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inputs</strong></td>
<td>9</td>
<td>25</td>
</tr>
<tr>
<td><strong>Outputs</strong></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Generations</strong></td>
<td>$3 \times 10^6$ (3 hours)</td>
<td>$3 \times 10^5$ (3 hours)</td>
</tr>
<tr>
<td><strong>Training vectors</strong></td>
<td>$1 \times 10^4$</td>
<td>$1 \times 10^5$</td>
</tr>
<tr>
<td><strong>Exact solution (K)</strong></td>
<td>38 operations</td>
<td>220 operations</td>
</tr>
<tr>
<td><strong>Available nodes (N)</strong></td>
<td>6 – 34 operations</td>
<td>10 – 200 operations</td>
</tr>
</tbody>
</table>

Approximate median: Distance error analysis

9-input median
fully-working: 38 operations

25-input median
fully-working: 220 operations

Evolutionary design of image filters from scratch

\[ \text{fitness} = \sum_{i=1}^{N} \sum_{j=1}^{M} |v(i, j) - w(i, j)| \]

Sekanina L. Image Filter Design with Evolvable Hardware. LNCS 2279, 2002
Comparison of approximate median filters and evolved filters for salt and pepper noise

PSNR – mean PSNR on 30 images
Synopsys Design compiler; 45 nm PDK
All filters are pipelined with $f_{\text{min}} = 1 \text{ GHz}$

Sekanina, Vasicek, Mrazek: Radioengineering 26(3), 2017
Testing of approximate circuits: New Challenges

- Instead of testing for all manufacturing defects, the goal is to test only for those that will lead to an error considered as not acceptable by the adopted Error Metrics.
- The main advantage: the test cost reduction

Fig. 1: AxIC Fault impact

Traiola M. et al.: On the Comparison of Different ATPG Approaches for Approximate Integrated Circuits. IEEE DDECS 2018
Incorrect subspace: The subset of input vectors for which the correct circuit and approximate circuit produce different outputs.

F (under-approximation):
Incorrect subspace is a subset of the on-set. 1 → 0 errors are produced

H (over-approximation)
Incorrect subspace is a subset of the off-set. 0 → 1 errors are produced

At most one of the circuits is allowed to produce an incorrect output for any input vector.

Approximate computing is a hot topic!

It is important as it addresses one of the most critical challenges of our society -- energy efficiency.

The roots of approximate computing
- the need for low power & high performance computing
- high fabrication variability in current/future technology nodes
- many dominant applications are error resilient

This tutorial is focused on approximate circuits and design methodologies.
- ad hoc circuit approximations
- automated circuit approximation methods
• A lot of research is needed
  • reliable quality (error) analysis methods -> Part II
  • benchmarking methodology
  • automated approximation methodologies
  • scalable and systematic solutions for complex systems approximation across the computer stack

• What would be the impact of the approximation on:
  • test
  • reliability
  • security
  • trust?
References

- Selected tutorial and survey papers on Approximate Computing
  - J. Han and M. Orshansky, “Approximate computing: An emerging paradigm for energy-efficient design,” in Proc. of the 18th IEEE European Test Symposium. IEEE, 2013, pp. 1–6
Acknowledgments

• Collaborators at FIT – Evolvable hardware group
  • Zdeněk Vašíček, Michal Bidlo, Vojtěch Mrázek, Jakub Husa, David Grochol, Michal Wiglasz
  • Former PhD students: Radek Hrbáček, Michaela Drahošová, Miloš Minařík

• Projects
  • IT4Innovations Centre of Excellence – National supercomputing center
  • Czech science foundation
    • Relaxed equivalence checking for approximate computing, 2016 – 2018
    • Advancing cryptanalytic methods through evolutionary computing, 2016 - 2018
    • Natural Computing on Unconventional Platforms, 2010 – 2013
  • Brno University Technology
Thank you for your attention!