An Automated, Efficient and Static Bit-width Optimization Methodology
Towards Maximum Bit-width-to-Error Tradeoff with Affine Arithmetic Model

Yu Pu ¹,²
Pu_Yu@nus.edu.sg

Yajun Ha ¹
elehy@nus.edu.sg

¹ National University of Singapore
² Technische Universiteit Eindhoven
Outline

- Introduction
- AA Bit-width analysis methodology
- Experimental results and verification
- Summary & Conclusions
Outline

- Introduction
- AA Bit-width analysis methodology
- Experimental results and verification
- Summary & Conclusions
Bit-width analysis is important!

- Bit-width analysis guarantees the precision of computation results.
- Bit-width analysis trades-offs precision with hardware resources, such as silicon area, power, etc.
We focus on static bit-width analysis
Simulation based analysis vs. Static analysis

Simulation based analysis
- Monte-Carlo style simulation, iterative trial
- Close to optimal bit-width results
- Often huge searching space with full coverage of input vectors, thus running time inefficient

Static analysis
- Infer bit-width for integer by value range propagation (forward, backward propagation)
- Infer bit-width for fraction by precision analysis
- Sub-optimal bit-width results
- Much shorter running time and iteration reduced
Interval Arithmetic overestimates bit-width

- Existing static bit-width analysis methods are IA based.
- What is Interval Arithmetic (IA)?

An uncertain variable $\overline{x}$ is expressed as $\overline{x} = [x_{\min}, x_{\max}]$

Take addition for an example:

$\overline{z} = \overline{x} + \overline{y} = [x_{\min} + y_{\min}, x_{\max} + y_{\max}]$

- An Example

Program

```
b = a - a;
```

Where a is 16-bit long

Interval Expressions

$b = [a_{\min} - a_{\max}, a_{\max} + a_{\max}]$

Results:

$b$ is 17-bit long

- IA overestimates bit-width, enabling fairly pessimistic results.
Affine Arithmetic estimates bit-width better

- **What is Affine Arithmetic (AA)**

  An uncertain variable \( \bar{x} \) is expressed as

  \[
  \hat{x} = x_0 + x_1 \varepsilon_1 + x_2 \varepsilon_2 + \cdots + x_n \varepsilon_n, -1 \leq \varepsilon_i \leq 1
  \]

  central value  coefficient  noise symbol

- **Previous Example**

  **Program**
  
  \[
  b = a - a;
  \]
  
  a is 16-bit long

  **Affine Expressions**
  
  \[
  a = a_0 + a_1 \varepsilon_a
  \]
  
  \[
  b = (a_0 + a_1 \varepsilon_a) - (a_0 + a_1 \varepsilon_a); = 0;
  \]

  **Results:**
  
  b is 1-bit long

- **AA models correlation between variables, enables tighter range propagation through cancelling some uncertainties along data-path.**
We extend AA for bit-width analysis

- Fang et. al. (CMU) have introduced AA model into the verification of the finite-precision effects in DSP applications. Detailed mathematical reasoning has been explained in [1] [2].
- We extend AA model for bit-width analysis.


Closely related work

A similar research was conducted by Dr. Lee [3] (imperial college). They applied AA together with the adaptive simulated annealing algorithm to find the optimal fractional bits.

Outline

- Introduction
- AA Bit-width analysis methodology
- Experimental results and verification
- Summary & Conclusions
Methodology overview

User interface

IA to AA

“Symbolically” forward propagation

Range analysis

Integer bit-width

Probabilistic error analysis

Hard error analysis

Probabilistic error analysis

Fractional bit-width

N

Y
User interface for MISO system
Methodology overview

User interface

IA to AA

“Symbolically” forward propagation

Convert input variables from IA form to AA form, introduce noise symbols

Range analysis

Integer bit-width

Probabilistic error analysis

Probabilistic error analysis

Hard error analysis

Fractional bit-width

Y

N
Methodology overview

User interface

IA to AA

“Symbolically” forward propagation

Traverse the whole program from entry to exit in forward direction

Range analysis

Probabilistic error analysis

Integer bit-width

Probabilistic error analysis

Fractional bit-width

Hard error analysis
Variable expression

Assume that there are totally $N$ floating-point variables whose upper quantization error bounds are expressed as

$$\left| \Delta_1 \right|, \left| \Delta_2 \right|, \left| \Delta_3 \right|, \cdots, \left| \Delta_N \right|$$

When they are transformed to fixed-point, the actual quantization errors incurred at each of them are expressed in affine forms as

$$\left| \Delta_1 \right| \varepsilon_1, \left| \Delta_2 \right| \varepsilon_2, \left| \Delta_3 \right| \varepsilon_3, \cdots, \left| \Delta_N \right| \varepsilon_N$$

The intermediate and output variable is expressed as

$$Variable = Const + \Delta Variable = Const + \left| \Delta_1 \right| \varepsilon_1 + \left| \Delta_2 \right| \varepsilon_2 + \left| \Delta_3 \right| \varepsilon_3 + \cdots + \left| \Delta_N \right| \varepsilon_N$$

central value error term
Methodology overview

User interface

IA to AA

“Symbolically” forward propagation

Range analysis

Integer bit-width

Probabilistic error analysis

Probabilistic error analysis

Hard error analysis

Fractional bit-width

Y

N
Methodology overview

User interface

IA to AA

“Symbolically” forward propagation

Range analysis

Integer bit-width

Probabilistic error analysis

Probabilistic error analysis

Hard error analysis

Fractional bit-width
Methodology overview

User interface

IA to AA

“Symbolically” forward propagation

Range analysis

Integer bit-width

Probabilistic error analysis

Probabilistic error analysis

Fractional bit-width

Hard error analysis

Y

N
Hard error analysis

The designer-specified output error tolerance $E_{\text{spec}}$ sets the upper bound of $\Delta_{\text{output}}$

$$|A_1|\Delta_1\varepsilon_1 + |A_2|\Delta_2\varepsilon_2 + |A_3|\Delta_3\varepsilon_3 + \cdots + |A_N|\Delta_N\varepsilon_N = \Delta_{\text{output}} \leq E_{\text{spec}}$$

To fully insure the output error not to exceed the designer-specified error tolerance, we take the extreme scenario, i.e.,

$$\varepsilon_1 = \varepsilon_2 = \varepsilon_3 = \cdots = \varepsilon_N = 1 \quad \text{Therefore, we have}$$

$$|A_1|\Delta_1 + |A_2|\Delta_2 + |A_3|\Delta_3 + \cdots + |A_N|\Delta_N \leq E_{\text{spec}}$$

Assign non-negative weights to each term of the equation,

$$\sum_{i=1}^{N} W_i |A_i|\Delta_i = E_{\text{spec}}$$

$$\sum_{i=1}^{N} W_i = 1$$

In our research, we assign each term with same weights $W_i = 1/N$
Methodology overview

User interface

IA to AA

“Symbolically” forward propagation

Range analysis

Integer bit-width

Probabilistic error analysis

Y

Probabilistic error analysis

Fractional bit-width

Hard error analysis

N

21
Probabilistic error analysis

In most of the DSP designs, the designer allows certain degree of error rate, which enables a larger bit-width-to-error tradeoff than using hard error analysis. Our probabilistic error analysis almost insures that the probability for the output error to lie in the specified error tolerance is higher than the designer specified parameter $\lambda$

$$\left| A_1 \right| \Delta_1 \varepsilon_1 + \left| A_2 \right| \Delta_2 \varepsilon_2 + \left| A_3 \right| \Delta_3 \varepsilon_3 + \cdots + \left| A_N \right| \Delta_N \varepsilon_N = \Delta_{\text{output}} \leq \text{Espec}$$

Let $A_i = A_i$, the equations depicts a sum of many statistically independent and identical distributed terms. By the central limit theorem, the equation approaches a Gaussian CDF.

$$\frac{\Delta_{\text{output}}}{\sqrt{N \text{ Variance}}} \rightarrow N(0,1)$$

$\text{Variance} = \frac{k^2}{3} = \left( \left| A_i \right| \Delta_i \right)^2 / 3$

$$\text{prob}(\Delta_{\text{output}} \leq E_{\text{spec}}) \geq \lambda$$

$\text{f}_i$
Outline

- Introduction
- Motivation
- AA Bit-width analysis methodology
- Experimental results and verification
- Summary & Conclusions
Experimental Results

A butterfly part in IDCT

\[ X_1 \in [-128,127] \quad X_2 \in [-128,127] \]

Error tolerance at \( Y_2 \) is 1.0

In probabilistic error analysis, the probability \( \lambda = 0.999 \).

![Diagram of butterfly part in IDCT]

<table>
<thead>
<tr>
<th>Node</th>
<th>IA</th>
<th>AA</th>
<th>IA</th>
<th>AA</th>
<th>IA</th>
<th>AA</th>
<th>IA</th>
<th>AA</th>
<th>IA</th>
<th>AA</th>
<th>IA</th>
<th>AA</th>
<th>IA</th>
<th>AA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Fractional</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>10</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

\begin{align*}
\text{Prob.} & = 2^{24}
\end{align*}
Experimental Results (Continued)

- Bit-width is normalized with respect to the IA based fractional bit-width.
- Hard error analysis → more than 30% fractional bit-width reduction
- Probabilistic error analysis → 50% reduction
- Tradeoff goes up as the relaxing the probabilistic restriction
Result Verification

- For the hard error analysis, whether the maximum output error lies within the specified tolerance after implementation?
- For the probabilistic error analysis, whether the probability for the output to lie in the specified error tolerance can be higher than the specified probability parameter?

Program in high level language C

Random input

Bit-width

Fixed-point quantization: real rounding

Linking with SystemC library

Output I

Output II
## Result Comparison and Analysis

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Bit-width Analysis Method</th>
<th>Simulated output error</th>
<th>Specified Probability</th>
<th>Simulated Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example I</td>
<td>IA based</td>
<td>0.02</td>
<td>N.A.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>AA based</td>
<td>0.22</td>
<td>N.A.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.47</td>
<td>0.99</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.33</td>
<td>0.999</td>
<td>1</td>
</tr>
<tr>
<td>FIR filter</td>
<td>IA based</td>
<td>0.09</td>
<td>N.A.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>AA based</td>
<td>0.15</td>
<td>N.A.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.58</td>
<td>0.99</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.58</td>
<td>0.999</td>
<td>1</td>
</tr>
<tr>
<td>4th order polynomial</td>
<td>IA based</td>
<td>0.88</td>
<td>N.A.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>AA based</td>
<td>0.93</td>
<td>N.A.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.10</td>
<td>0.99</td>
<td>0.999</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.95</td>
<td>0.999</td>
<td>1</td>
</tr>
</tbody>
</table>
Outline

- Introduction
- Motivation
- AA Bit-width analysis methodology
- Experimental results and verification
- Summary & Conclusions
Summary & Conclusions

- An automated and efficient static bit-width analysis methodology based on AA model is presented.
- The proposed probabilistic error analysis can further shorten the bit-width and explore a larger bit-width-to-error tradeoff.

Limitations:
- Our method lies in algorithm level which does not consider the hardware cost function.
- We use Gaussian approximation during analysis, theoretically it is difficult to fully guarantee the error probability to be bounded. Therefore, we suggest that the specified probability should be flexibly restricted.
Thank you!