Efficient Analysis of Multidimensional Linear Systems for Wordlength Optimization

Gaël Deest    Tomofumi Yuki    Olivier Sentieys
Steven Derrien

This work was funded by European FP7 project Alma
Embedded System Design

Many constraints:

• Power efficiency
• Production cost
• Performance / speed
• Time-to-market
• …
Design-Space Exploration (DSE)

Cost = power or area
Design-Space Exploration (DSE)

Cost = power or area

Time constraint

Execution time
Design-Space Exploration (DSE)

Cost

Time constraint

Execution time

Cost = power or area

Optimum
Design-Space Exploration (DSE)

Cost = power or area

Accuracy degradation (Signal to Noise Ratio)
Design-Space Exploration (DSE)

Cost = power or area

Custom **fixed-point formats** used to reduce cost
Wordlength Optimization Process

Soft accuracy constraints (eg., noise power)

Fast accuracy evaluation is critical for thorough design-space exploration
This Work

State of the art:

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Applicability</th>
<th>Depth of DSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation-based</td>
<td>Excellent</td>
<td>Limited</td>
</tr>
<tr>
<td>Current analytical</td>
<td>Limited</td>
<td>Good</td>
</tr>
<tr>
<td>techniques</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Our approach</strong></td>
<td><strong>Good</strong></td>
<td><strong>Good</strong></td>
</tr>
</tbody>
</table>

**Diagnostic:** Applicability issues for analytical techniques.
**Contribution:** Extended applicability and scalability.
Overview

• Background
• Analytical techniques
• Proposed approach
Fixed-Point Arithmetic

• Scaled integers:

\[ 2^{-k} \times \text{Integer value} \]

• Product of 2 \( n \)-bit numbers \( \rightarrow 2n \) bits!

• Some bits must be dropped (quantization)

Example (truncation):

\[
\begin{array}{cccccccc}
2^3 & 2^2 & 2^1 & 2^0 & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} & 2^{-5} & 2^{-6} \\
\end{array}
\]

\[ \text{Dropped bits} \]
Quantization Errors

Modeled as **noise / random variable**

**Example:** Truncation to $2^{-n}$ precision

$error \sim U([-2^{-n}; 0])$

- Assumptions: *Widrow hypothesis*
- Statistical moments:

  \[ \mu = -2^{-n-1} \quad \sigma^2 = \frac{2^{-2n}}{12} \]
Analytical Techniques

**Goal:** Compute an **output noise formula**

**Idea:** Model propagation of errors to the output

**Representation:** Signal Flow Graphs (SFG)
Accuracy Model Construction
Accuracy Model Construction

Quantization errors = new inputs
Accuracy Model Construction

Quantization errors = new inputs

\[ x(n) \xrightarrow{+} X \xrightarrow{+} \text{err} \]

\[ \text{b0} \xrightarrow{+} \text{X} \xrightarrow{+} \text{y}(n) \]

\[ \text{D} \xrightarrow{b1} \text{X} \xrightarrow{+} \text{D} \]

\[ \text{b2} \xrightarrow{+} \text{X} \xrightarrow{+} \text{D} \]

\[ \text{a1} \xrightarrow{+} \text{X} \xrightarrow{+} \text{D} \]

\[ \text{a2} \xrightarrow{+} \text{X} \xrightarrow{+} \text{D} \]
Accuracy Model Construction

Compute **transfer function** for each error

\[ x(n) + b_0 \times X + b_1 \times D + b_2 \times X + D \times a_1 + D \times a_2 \times X \rightarrow y(n) \]
The Challenge

How to go from this...

```c
float xb[N];

float fir(float in) {
    float y = 0;
    xb[0] = in;

    for (int i=0; i<N; i++)
        acc += b[i]*xb[i];

    for (int i=N-1; i>0; i--)
        xb[i] = xb[i-1];

    return y;
}
```

...to this?

![Diagram of filter](image)
The Challenge

Current methods:
• Flatten control (completely unroll loops, etc.)
• Heavy use of annotations:
  **Example:**  #pragma DELAY
  float xb[N];

Limitations:
• Scalability issues (large graphs)
• Implicit 1D stream assumption
• Not easily applicable to image processing
Motivating Example: Deriche Filter

Horizontal:

Recursive Filter

Vertical:

Similar along columns
Motivating Example: Deriche Filter

Issues with SFG representation:

- Requires image size to be statically known
- Each pixel is a different input
- Number of transfer functions: $O(N^4)$
  - For 32x32 image: 1,048,576!

Cannot be handled with current methods
Intuition of the Technique

• Current tools cannot capture regularity of multidimensional filters.

Idea:
• Generalize SFGs to multidimensional systems of equations.
• Infer this representation using polyhedral dependence analysis.
Steps of our Method

1. Build an **equational** representation of the program.

```c
float xb[N];

float fir(float x) {
    float y = 0;
    xb[0] = in;

    for (int i=0; i<N; i++)
        y += b[i]*xb[i];

    for (int i=N-1; i>0; i--)
        xb[i] = xb[i-1];

    return y;
}
```

\[
\begin{align*}
S_0(n) &= 0 \\
S_1(n) &= x(n) \\
S_2(n, i) &= \begin{cases} 
S_0(n) + b(i) \times S_1(n) & i = 0 \\
S_2(n, i - 1) + b(i) \times S_3(n - 1, i) & i > 0 
\end{cases} \\
S_3(n, i) &= \begin{cases} 
S_1(n) & i = 1 \\
S_3(n - 1, i - 1) & i > 1 
\end{cases} \\
y(n) &= S_2(n, N - 1)
\end{align*}
\]
Equational Representation

Example:

```c
float tmp = 0;
for (int i=0; i<N; i++)
    tmp = arr[i] + tmp;
```

\[
S_0() = 0
\]
\[
S_1(i) = \begin{cases} 
S_0 & i = 0 \\
S_1(i-1) & i > 0 
\end{cases} + \text{arr}(i)
\]

- Statement \( \equiv \) equation
- Keeps track of data dependencies
- Easy to transform / reason about
- Relies on Array Dataflow Analysis (Feautrier, 1991)
Example: Simplified Deriche Filter

```cpp
for (int i=0; i<N; i++) {
    prev = 0;
    for (int j=0; j<N; j++) {
        tmp[i][j] = a1*x[i][j] + b1*prev;
        prev = tmp[i][j];
    }
}
```

```
for (int j=0; j<N; j++) {
    prev = 0;
    for (int i=0; i<N; i++) {
        y[i][j] = a2*tmp[i][j] + b2*prev;
        prev = y[i][j];
    }
}
```

Horizontal pass (row scan)

Vertical pass (column scan)
Equation System

After pre-processing:

\[
\begin{align*}
S_1(i, j) &= a_1 x(i, j) + b_1 S_1(i, j - 1) \\
S_2(j, i) &= a_2 S_1(i, j) + b_2 S_2(j, i - 1)
\end{align*}
\]
Equation System

After pre-processing:

\[
\begin{align*}
S_1(i, j) &= a_1 x(i, j) + b_1 S_1(i, j - 1) \\
S_2(j, i) &= a_2 S_1(i, j) + b_2 S_2(j, i - 1)
\end{align*}
\]

Swapped dimensions
(Non-uniform dependences)
Steps of our method

2. Uniformization

\[
\begin{align*}
    S_1(i, j) &= a_1 x(i, j) + b_1 S_1(i, j - 1) \\
    S_2(i, j) &= a_2 S_1(i, j) + b_2 S_2(i - 1, j)
\end{align*}
\]
Steps of our Method

3. Convolution Detection

Computation pattern:

\[ y(k) = \sum_{v} c(v) \times x(k - v) \]

- Pattern matching.
- Simplifies noise propagation.
Convolution Detection

After pre-processing:

\[
\begin{align*}
S_1(i, j) &= a_1 x(i, j) + b_1 S_1(i, j - 1) \\
S_2(i, j) &= a_2 S_1(i, j) + b_2 S_2(i - 1, j)
\end{align*}
\]
Convolution Detection

After pre-processing:

\[
\begin{align*}
S_1(i, j) &= a_1 x(i, j) + b_1 S_1(i, j - 1) \\
S_2(i, j) &= a_2 S_1(i, j) + b_2 S_2(i - 1, j)
\end{align*}
\]
Accuracy Model Construction

4. Compute noise propagation for each source

Extract subfilter to the output.

**Example:** From statement $S_1$ to $S_2$
Impulse Response Computation

Determines noise propagation:

\[ \text{err}_{\text{out}}(\nu) = (\text{err}_{\text{in}} * h)(\nu) \]

- Easy to compute for non-recursive filters
- Infinite for recursive filters
Non-Recursive Filters

\[
\begin{align*}
  z &= x \ast h_1 + x \ast h_2 \\
  y &= z \ast h_3
\end{align*}
\]
Non-Recursive Filters

\[
\begin{align*}
z &= x \ast h_1 + x \ast h_2 \\
y &= z \ast h_3
\end{align*}
\]
Non-Recursive Filters

\[
\begin{align*}
z &= x \ast (h_1 + h_2) \\
y &= z \ast h_3
\end{align*}
\]
Non-Recursive Filters

\[
\begin{align*}
z &= x \times (h_1 + h_2) \\
y &= z \times h_3
\end{align*}
\]
Non-Recursive Filters

\[
y = x \ast (h_3 \ast (h_1 + h_2))
\]

\[
h = h_3 \ast (h_1 + h_2)
\]
Recursive Filters

\[ y = x \ast h_1 + y \ast h_2 \]

- Finding \( h \equiv \) solving multidimensional recurrence
- Hard problem
Impulse Response Approximation

• **Hypothesis:** Filter is stable

\[ \sum_\nu |h(\nu)| < \infty \]

• **Consequence:**

\[ \lim_{r \to \infty} \sum_{|\nu| > r} h(\nu) = 0 \]

• **Idea:** Evaluate impulse response on sufficiently large window.
Back to the Definition

• Impulse response = output of the filter when the input is a *unit impulse*:

\[
\delta(v) = \begin{cases} 
1 & v = 0 \\
0 & \text{otherwise}
\end{cases}
\]

• Feed the filter with impulse and use the output as impulse response
### Experimental Results: Model Construction Time

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>ID.Fix (s)</th>
<th>Our Tool (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIR8</td>
<td>23.1</td>
<td>20.5</td>
</tr>
<tr>
<td>Sobel (32 × 32)</td>
<td>169.1</td>
<td>9.2</td>
</tr>
<tr>
<td>Sobel (64 × 64)</td>
<td>2173.1</td>
<td>9.7</td>
</tr>
<tr>
<td>Sobel (128 × 128)</td>
<td>-</td>
<td>9.4</td>
</tr>
<tr>
<td>Gaussian blur (32 × 32)</td>
<td>160.1</td>
<td>10.2</td>
</tr>
<tr>
<td>Gaussian blur (64 × 64)</td>
<td>2010.9</td>
<td>9.5</td>
</tr>
<tr>
<td>Gaussian blur (128 × 128)</td>
<td>-</td>
<td>9.4</td>
</tr>
<tr>
<td>Deriche (16 × 16)</td>
<td>-</td>
<td>6.5</td>
</tr>
</tbody>
</table>
## Experimental Results: Model Validity

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Simulation (dB)</th>
<th>Our Tool (dB)</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIR8</td>
<td>-17.80</td>
<td>-17.84</td>
<td>-0.2</td>
</tr>
<tr>
<td>Sobel</td>
<td>11.62</td>
<td>12.04</td>
<td>3.6</td>
</tr>
<tr>
<td>Gauss</td>
<td>3.78</td>
<td>3.78</td>
<td>0.1</td>
</tr>
<tr>
<td>Deriche</td>
<td>-18.01</td>
<td>-18.06</td>
<td>-2.78</td>
</tr>
</tbody>
</table>
Conclusion

1. **Extraction** of a **compact** program representation (generalization of SFGs).

2. **Reformulation** of analytical techniques on this representation.

3. **Wider applicability** for analytical accuracy analysis
Open Issues

• Extension to non linear, non time-invariant filters
  • Extensions exist for 1D SFGs
  • Expected to be easily applicable to our model
• Regular, but non affine programs
  • Example: FFT
• Highly correlated Inputs