Adaptive Mapping of Linear DSP Algorithms to Fixed-Point Arithmetic

Lawrence J. Chang
Inpyo Hong
Yevgen Voronenko
Markus Püschel

Department of Electrical & Computer Engineering
Carnegie Mellon University

Supported by NSF awards ACR-0234293, SYS-0310941, and ITR/NGS-0325687
Motivation

- Embedded DSP applications (SW and HW) typically use fixed-point arithmetic for reduced power/area and better throughput.

- Typically DSP algorithms are manually mapped to fixed-point implementation.
  - time consuming, non-trivial task
  - difficult trade-off between range (to avoid overflow) and precision
  - usually done using simulations (not an exact science)

- Our goal: automatically generate overflow-proof, and accurate fixed-point code (SW) for linear DSP kernels using the SPIRAL code generator.
Outline

- Background
- Approach using SPIRAL
  - Mapping to Fixed Point Code (Affine Arithmetic)
  - Accuracy Measure
- Probabilistic Analysis
- Results
Background: SPIRAL

- Generates fast, platform-adapted code for linear DSP transforms (DFT, DCTs, DSTs, filters, DWT, …)
- Adapts by searching in the algorithm space and implementation space for the best match to the platform
- Floating-point code only
- Our goal: extend SPIRAL to generate overflow-proof, accurate fixed-point code

www.spiral.net
Background: Transform Algorithms

- Reduce computation cost from $O(n^2)$ to $O(n \log n)$ or below
- For every transform there are many algorithms
- An algorithm can be represented as
  - Sparse matrix factorization
  \[
  \begin{pmatrix}
  y_0 \\
  y_1 \\
  y_2 \\
  y_3
  \end{pmatrix} =
  \begin{pmatrix}
  1 & -1 & & \\
  & 1 & -1 & \\
  1 & 1 & & \\
  1 & 1 & & 
  \end{pmatrix}
  \begin{pmatrix}
  1 & 1 & 1 & \\
  a & c & s & \\
  -s & c & & \\
  & & & 
  \end{pmatrix}
  \begin{pmatrix}
  x_0 \\
  x_1 \\
  x_2 \\
  x_3
  \end{pmatrix}
  \]
- Data flow DAG (Directed Acyclic Graph)
- Program
  \[
  \begin{align*}
  t_1 &= a \times x_2 \\
  t_2 &= t_1 + x_0 \\
  t_3 &= -s \times x_1 + c \times x_3 \\
  y_3 &= t_2 + t_3 \\
  y_0 &= t_2 - t_3 \\
  \ldots & \ldots \\
  \end{align*}
  \]
Background: Fixed-Point Arithmetic

- Uses integers to represent fractional numbers:
  
  ![Fixed-Point Representation Diagram]

  - sign
  - integer bits
  - fractional bits

  register width: $RW = 1 + IB + FB$ (typically 16 or 32)

- Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a+b$</td>
<td>$a \cdot b \gg fb$</td>
</tr>
</tbody>
</table>

  addition           multiplication

- Dynamic range:
  - $-2^{IB} \ldots 2^{IB}-1$
  - much smaller than in floating-point) risk of overflow

- Problem: for a given application, choose $IB$ (and thus $FB$) to avoid overflow

- We present an algorithm to automatically choose, application dependent, “best” $IB$ (and thus $FB$) for linear DSP kernels

Example (RW=9, IB=FB=4)

$0011\ 0011_2 = 1011.0111_2 = 3.1875_{10}$
Outline

- Background
- Approach using SPIRAL
  - Mapping to Fixed Point Code (Affine Arithmetic)
  - Accuracy Measure
- Probabilistic Analysis
- Results
Overview of Approach

- Extension of SPIRAL code generator
- **Fixed-point mapping**: maps floating-point code into fixed-point code, given the input range
- Use SPIRAL to **automatically** search for the fixed-point implementation
  - with highest accuracy, or
  - with fastest runtime
Tool: Affine Arithmetic

- Basic idea: propagate ranges through the computation (interval arithmetic, IA); each variable becomes an interval
- Problem: leads to range overestimation, since correlations between variables are not considered
- Solution: affine arithmetic (AA) [1]
  - represents range as affine expression
  - captures correlations

IA: \( A(x) = [-M,M] \)
AA: \( A(x) = c_0 \cdot E_0 + c_1 \cdot E_1 + \ldots \)

\( E_i \) are ranges, e.g., \( E_i = [-1,1] \)

[1] Fang Fang, Rob A. Rutenbar, Markus Püschel, and Tsuhan Chen
Toward Efficient Static Analysis of Finite-Precision Effects in DSP Applications via Affine Arithmetic Modeling
Proc. DAC 2003, pp. 496-501
Algorithm 1 [Range Propagation]

- **Input:** Program with additions and multiplications by constants, ranges of inputs
- **Output:** Ranges of outputs and intermediate results

- Denote input ranges by $x_i$ with $i \in \{1, N\}$
- We represent all variables $v$ as affine expressions $A$:
  \[ A(v) = \sum_{i=0}^{n-1} c_i \cdot x_i \quad \text{where } c_i \text{ are constants} \]

- Traverse all variables from input to output, and compute $A$:
  \[ A(x_i) = x_i \]
  \[ A(v_1 + v_2) = A(v_1) + A(v_2) \]
  \[ A(c \cdot v) = c \cdot A(v) \]

- Variable ranges $R = [R_{\text{min}}, R_{\text{max}}]$ are given by
  \[ R_{\text{min}}(A(v)) = R_{\text{min}}(\sum_i c_i \cdot x_i) = \sum_i |c_i| \cdot R_{\text{min}}(x_i) \]
  \[ R_{\text{max}}(A(v)) = R_{\text{max}}(\sum_i c_i \cdot x_i) = \sum_i |c_i| \cdot R_{\text{max}}(x_i) \]
Example

Program
\[ t_1 = x_1 + x_2 \]
\[ t_2 = x_1 - x_2 \]
\[ y_1 = 1.2 \times t_1 \]
\[ y_2 = -2.3 \times t_2 \]
\[ y_3 = y_1 + y_2 \]

Affine Expressions
\[ A(t_1) = x_1 + x_2 \]
\[ A(t_2) = x_1 - x_2 \]
\[ A(y_1) = 1.2 \times x_1 + 1.2 \times x_2 \]
\[ A(y_2) = -2.3 \times x_1 + 2.3 \times x_2 \]
\[ A(y_3) = -1.1 \times x_1 + 3.5 \times x_2 \]

Computed Ranges
\[ R(t_1) = [-2,2] \]
\[ R(t_2) = [-2,2] \]
\[ R(y_1) = [-2.4,2.4] \]
\[ R(y_2) = [-2.6,2.6] \]
\[ R(y_3) = [-4.6,4.6] \]

Given Ranges
\[ R(x_1) = [-1,1] \]
\[ R(x_2) = [-1,1] \]

ranges are exact (not worst cases)
Algorithm 2 [Error Propagation]

- **Input:** Program with additions and multiplications by constants, ranges of inputs
- **Output:** Error bounds on outputs and intermediate results

- Denote by $\varepsilon_i$ in [-1,1] independent random error variables
- We augment affine expressions $A$ with error terms:
  \[ A_{\varepsilon}(v) = \sum_{i=0}^{n-1} c_i \cdot x_i + \sum_j f_j \cdot \varepsilon_j \]
  where $f_i$ are error magnitude constants

- Traverse all variables from input to output, and compute $A_{\varepsilon}$:
  \[
  A_{\varepsilon}(x_i) = x_i \\
  A_{\varepsilon}(v_1 + v_2) = A_{\varepsilon}(v_1) + A_{\varepsilon}(v_2) + 2^{-r_w} |R_{\text{max}}(v_1 + v_2)| \varepsilon \\
  A_{\varepsilon}(c \cdot v) = c \cdot A_{\varepsilon}(v) + 2^{-r_w} |R_{\text{max}}(c \cdot v)| \varepsilon
  \]

- Maximum error is given by
  \[ \mathcal{E}(v) = \sum_j |f_j| \]
Fixed-Point Mapping

- **Input:**
  - floating point program (straightline code) for linear transform
  - ranges of input

- **Output:** fixed-point program

- **Algorithm:**
  - Determine the affine expressions of all intermediate and output variables; compute their maximal ranges
  - **Mode 1: Global format**
    - the largest range determines the fixed point format globally
  - **Mode 2: Local format**
    - allow different formats for all intermediate and output variables
  - Convert floating-point constants into fixed-point constants
  - Convert floating-point operations into fixed-point operations
  - Output fixed-point code
**Accuracy Measure**

- **Goal:** evaluate a SPIRAL generated fixed-point program for accuracy to enable search for best = most accurate algorithm
- Choose input independent accuracy measure: matrix norm

\[ \| T - \hat{T} \|_\infty \quad \text{max row sum norm} \]

- matrix for exact (floating-point) program
- matrix for fixed-point program

**Note:** can be used to derive input dependent error bounds

\[ \| y - \hat{y} \|_\infty \leq \| T - \hat{T} \|_\infty \| x \|_\infty \]
Outline

- Background
- Approach using SPIRAL
  - Mapping to Fixed Point Code (Affine Arithmetic)
  - Accuracy Measure
- Probabilistic Analysis
- Results
Probabilistic Analysis

Fixed point mapping chooses range conservatively, namely:

\[ A(x) = c_0 x_0 + c_1 x_1 + \cdots \]

leads to a range estimate of

\[
\left[ \sum_i c_i \mid \min(|x_i|), \sum_i c_i \mid \max(|x_i|) \right]
\]

However: not all values in \([-M,M]\] are equally likely

Analysis:

- Assume \(x_i\) are uniformly distributed, independent random variables
- Use **Central Limit Theorem**: \(A(x)\) is approximately Gaussian
- Extend Fixed-Point Mapping to include a probabilistic mode (range satisfied with given probability \(p\))
Overestimation due to Central Limit Theorem

affine expression with:

- 4 terms
- 16 terms
- 64 terms

assuming input/error variables are independent
Outline

- Background
- Approach using SPIRAL
  - Mapping to Fixed Point Code (Affine Arithmetic)
  - Accuracy Measure
- Probabilistic Analysis
- Results
Accuracy Histogram

DCT, size 32
10,000 random algorithms
Spiral generated

- Spread 10x, most within 2x
- Need for search
Global vs. Local Mode

Local mode a factor of 1.5-2 better
Local vs. Gaussian Local Mode

Gain: about a factor of 2.5-4

99.99% confidence for each variable
Summary

- An automatic method to generate accurate, overflow-proof fixed-point code for linear DSP kernels
  - Using SPIRAL to find the most accurate algorithm: 2x
  - Floating-point to fixed-point using affine arithmetic analysis (global, local: 2x, probabilistic: 4x)
  - 16x

- Current work:
  - Extend approach to handle loop code and thus arbitrary size transforms
  - Refine probabilistic mode to get statements as:
    \[ \text{prob(overflow)} < p \]

- Further down the road:
  - Fixed-point mapping compiler for more general numerical DSP kernels/applications

www.spiral.net