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

Carnegie Mellon

SPIRAL

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

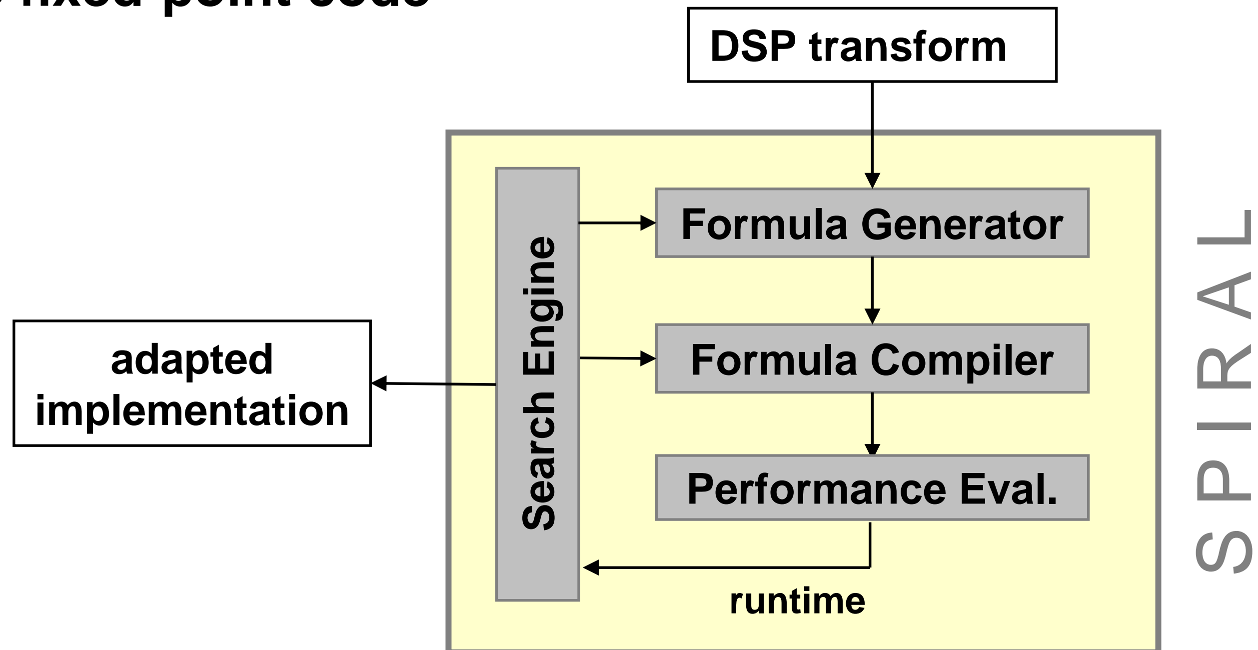
- Background
- Approach using SPIRAL
  - Mapping to Fixed Point Code (Affine Arithmetic)
  - Accuracy Measure
- Probabilistic Analysis
- Results

Carnegie Mellon

SPIRAL

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

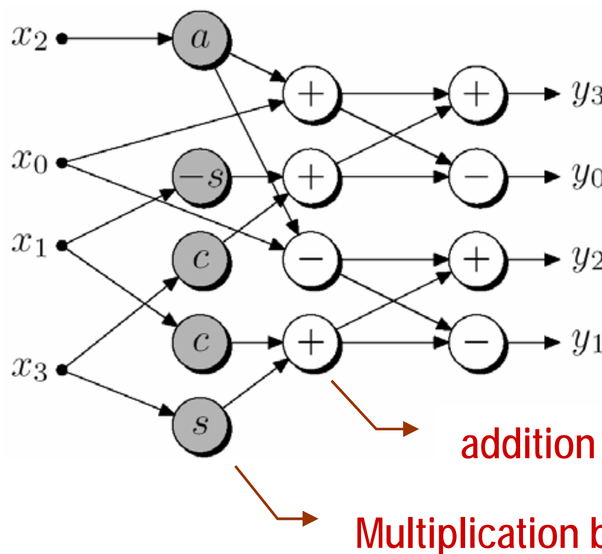


Background: Transform Algorithms

- 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

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 & -1 & & \\ & & 1 & -1 \\ & & 1 & 1 \\ 1 & 1 & & \end{pmatrix} \begin{pmatrix} 1 & 1 & & \\ & & & 1 \\ 1 & -1 & & \\ & & 1 & \end{pmatrix} \begin{pmatrix} 1 & & & \\ & & a & \\ & c & & s \\ & -s & & c \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

  - Data flow DAG (Directed Acyclic Graph)



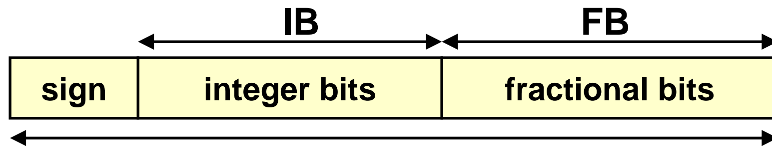
  - Program

```
t1 = a * x2
t2 = t1 + x0
t3 = -s * x1 + c * x3
y3 = t2 + t3
y0 = t2 - t3
… …
… …
```

Carnegie Mellon
SPIRAL

Background: Fixed-Point Arithmetic

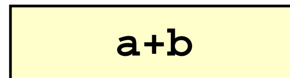
- Uses integers to represent fractional numbers:



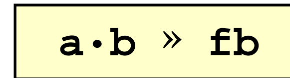
register width: RW = 1 + IB + FB (typically 16 or 32)

Example (RW=9, IB=FB=4)
$0011\ 0011_2 = 1011.0111_2 = 3.1875_{10}$

- Operations



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

Carnegie Mellon

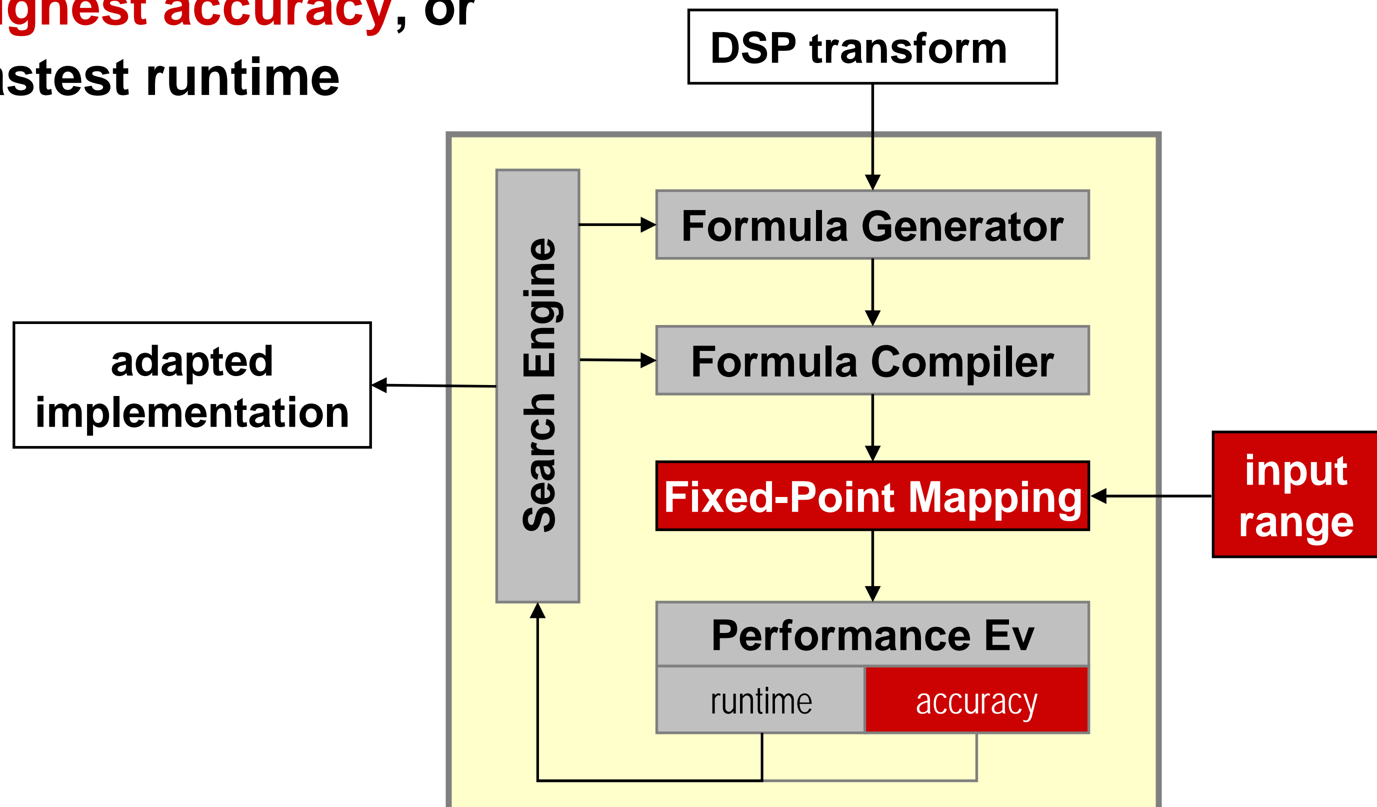
SPIRAL

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



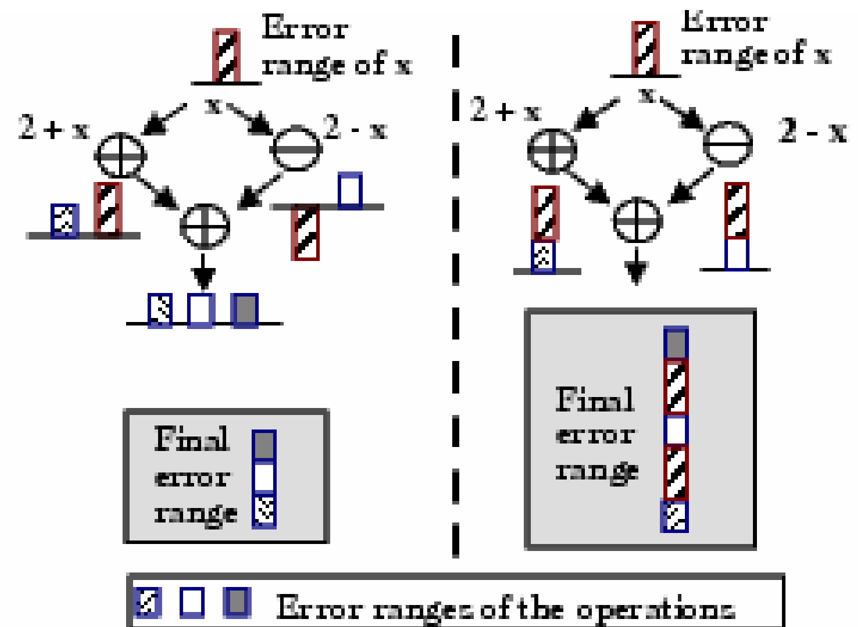
Carnegie Mellon
SPIRAL

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 i2 [1, N]**
  - **We represent all variables v as affine expressions A:**

$$\mathcal{A}(v) = \sum_{i=0}^{n-1} c_i \cdot x_i$$

where c_i are constants

  - **Traverse all variables from input to output, and compute A:**

$$\begin{aligned} \mathcal{A}(x_i) &= x_i \\ \mathcal{A}(v_1 + v_2) &= \mathcal{A}(v_1) + \mathcal{A}(v_2) \\ \mathcal{A}(c \cdot v) &= c \cdot \mathcal{A}(v) \end{aligned}$$

  - **Variable ranges R=[R_{min},R_{max}] are given by**

$$\mathcal{R}_{\text{min}}(\mathcal{A}(v)) = \mathcal{R}_{\text{min}}(\sum_i c_i \cdot x_i) = \sum_i |c_i| \cdot \mathcal{R}_{\text{min}}(x_i)$$

$$\mathcal{R}_{\text{max}}(\mathcal{A}(v)) = \mathcal{R}_{\text{max}}(\sum_i c_i \cdot x_i) = \sum_i |c_i| \cdot \mathcal{R}_{\text{max}}(x_i)$$

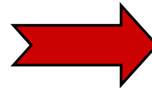
Carnegie Mellon

SPIRAL

Example

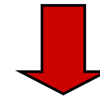
Program

```
t1 = x1 + x2
t2 = x1 - x2
y1 = 1.2 * t1
y2 = -2.3 * t2
y3 = y1 + y2
```



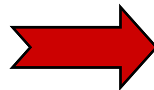
Affine Expressions

```
A(t1) = x1 + x2
A(t2) = x1 - x2
A(y1) = 1.2 x1 + 1.2 x2
A(y2) = -2.3 x1 + 2.3 x2
A(y3) = -1.1 x1 + 3.5 x2
```



Given Ranges

```
R(x1) = [-1,1]
R(x2) = [-1,1]
```



Computed Ranges

```
R(t1) = [-2,2]
R(t2) = [-2,2]
R(y1) = [-2.4,2.4]
R(y2) = [-2.6,2.6]
R(y3) = [-4.6,4.6]
```

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 ε_i in [-1,1] independent random error variables
  - We augment affine expressions A with error terms:

$$\mathcal{A}_\epsilon(v) = \sum_{i=0}^{n-1} c_i \cdot x_i + \sum_j f_j \cdot \epsilon_j$$

where f_i are error magnitude constants

  - Traverse all variables from input to output, and compute A_ε:

$$\begin{aligned} \mathcal{A}_\epsilon(x_i) &= x_i \\ \mathcal{A}_\epsilon(v_1 + v_2) &= \mathcal{A}_\epsilon(v_1) + \mathcal{A}_\epsilon(v_2) + \overbrace{2^{-rw}|\mathcal{R}_{\max}(v_1 + v_2)|}^{f}\epsilon \\ \mathcal{A}_\epsilon(c \cdot v) &= c \cdot \mathcal{A}_\epsilon(v) + 2^{-rw}|\mathcal{R}_{\max}(c \cdot v)|\epsilon \end{aligned}$$

new error variable introduced

  - 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

Carnegie Mellon
SPIRAL

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**

$$\| T - \hat{T} \|_\infty$$

max row sum norm

T: matrix for exact (floating-point) program

$\hat{T}$: matrix for fixed-point program

Note: can be used to derive input dependent error bounds

$$\| y - \hat{y} \|_\infty \le \| T - \hat{T} \|_\infty \| x \|_\infty$$

Carnegie Mellon
SPIRAL

Outline

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

Carnegie Mellon
SPIRAL

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

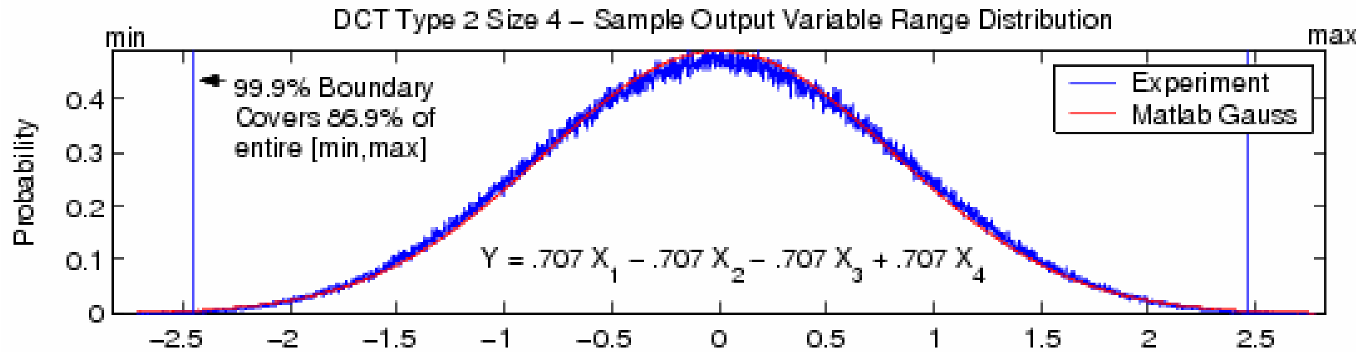
$$\left[\sum_i |c_i| \min(|x_i|), \sum_i |c_i| \max(|x_i|) \right]$$

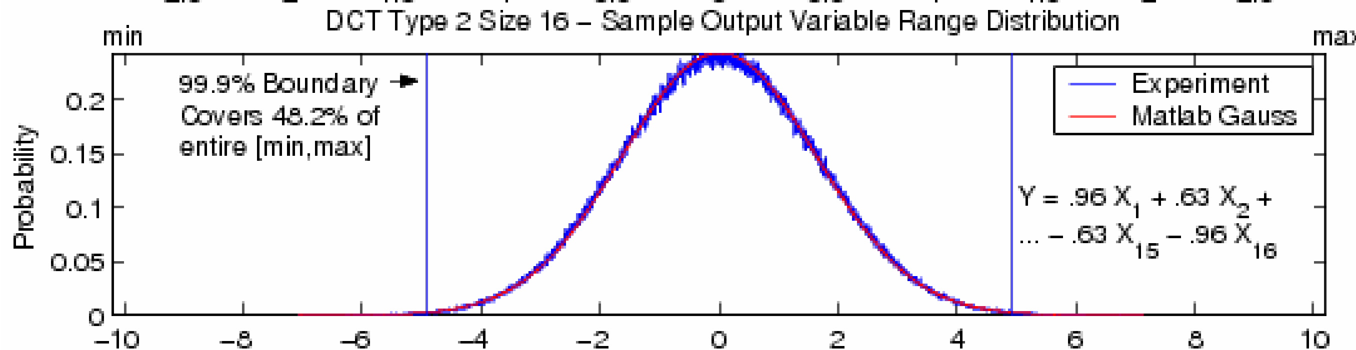
However: not all values in [-M,M] are equally likely

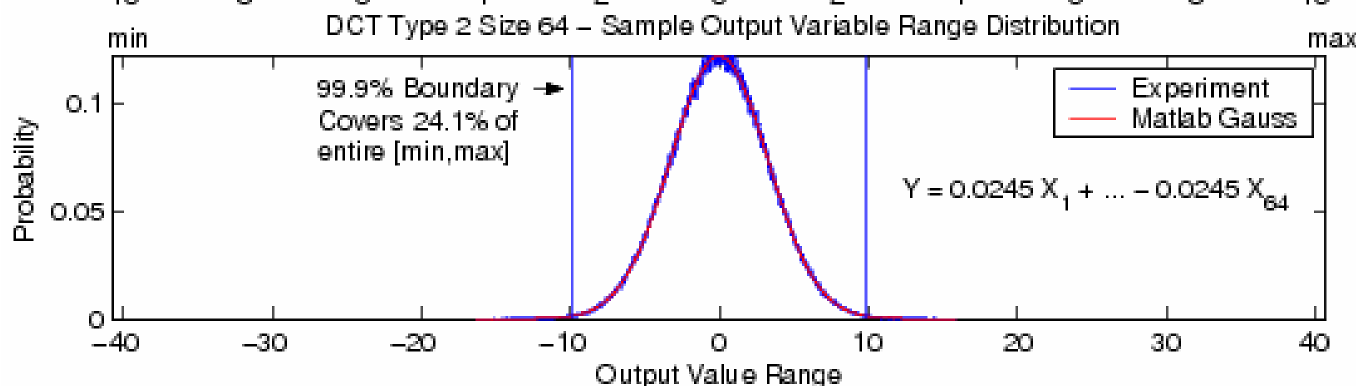
Analysis:

- Assume xi 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

Carnegie Mellon

SPIRAL

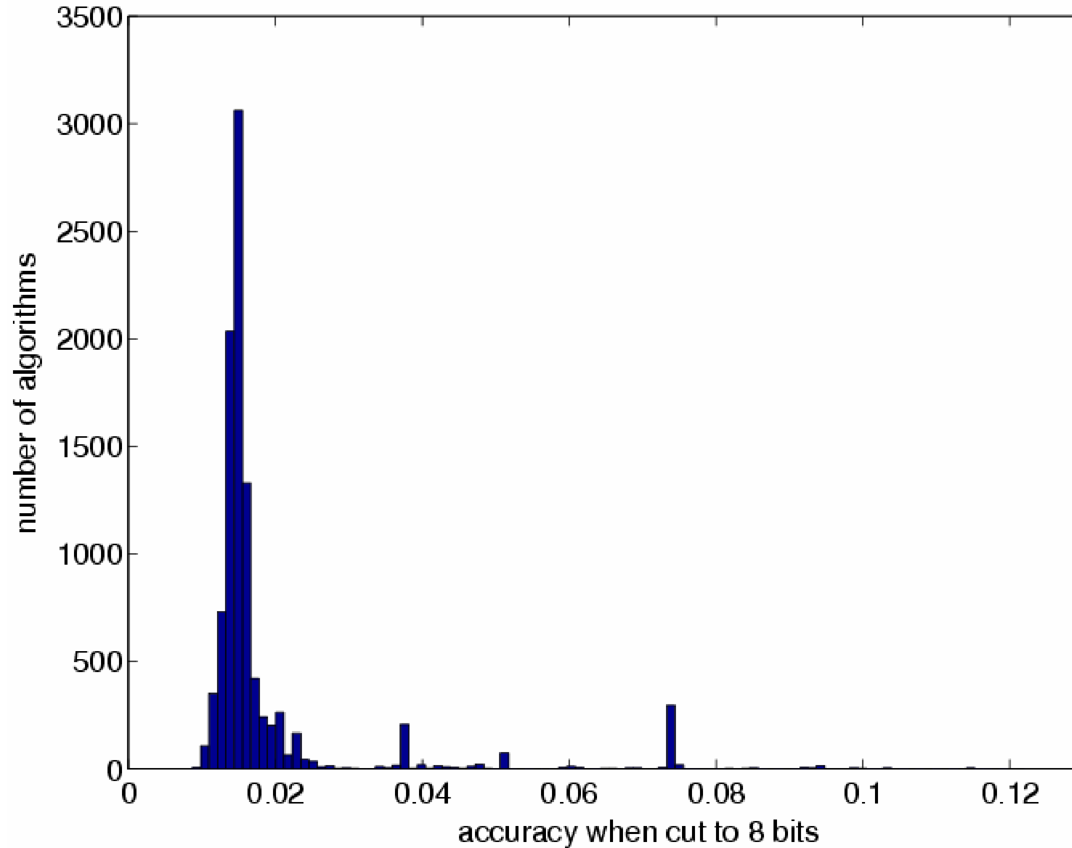
Outline

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

Carnegie Mellon
SPIRAL

Accuracy Histogram

DCT, size 32
10,000 random algorithms
Spiral generated

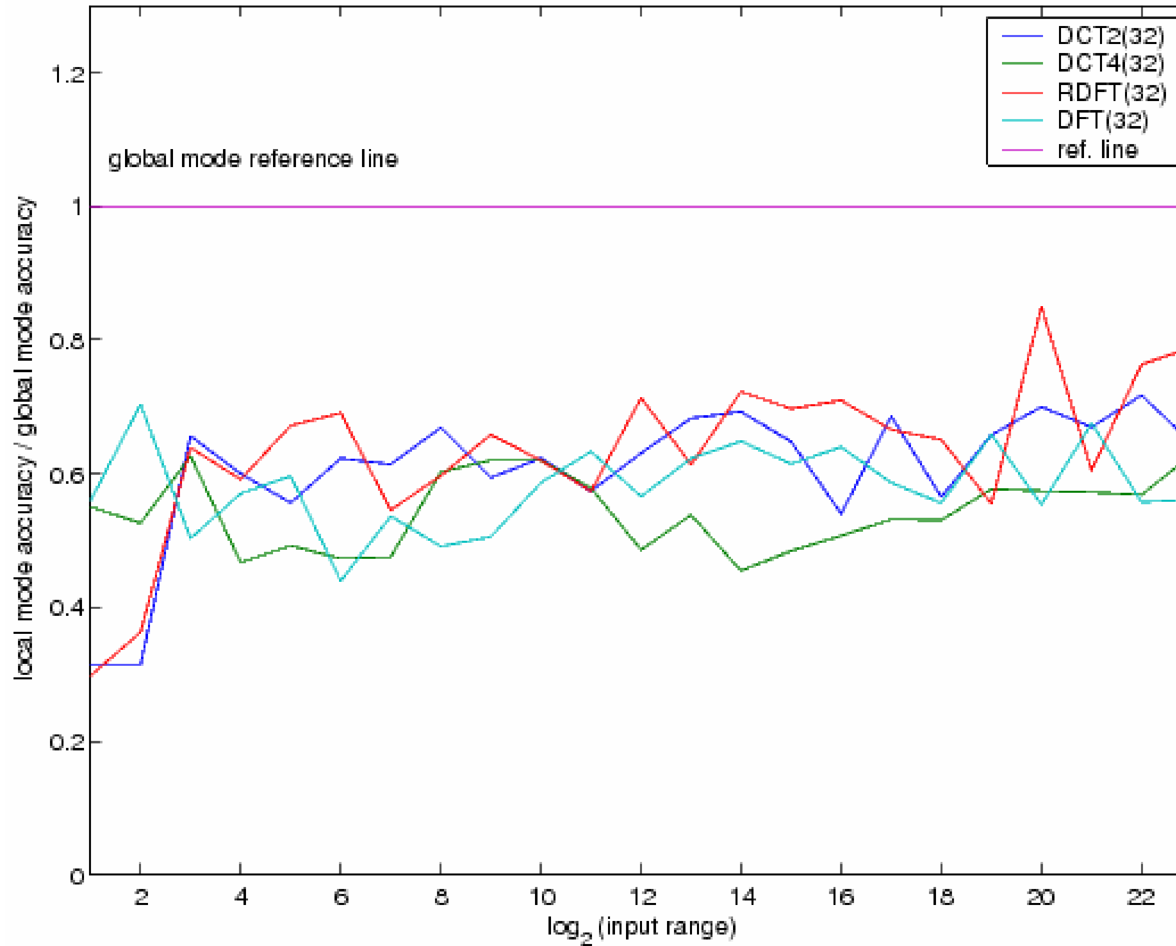


- **Spread 10x, most within 2x**
- **Need for search**

Carnegie Mellon

SPIRAL

Global vs. Local Mode



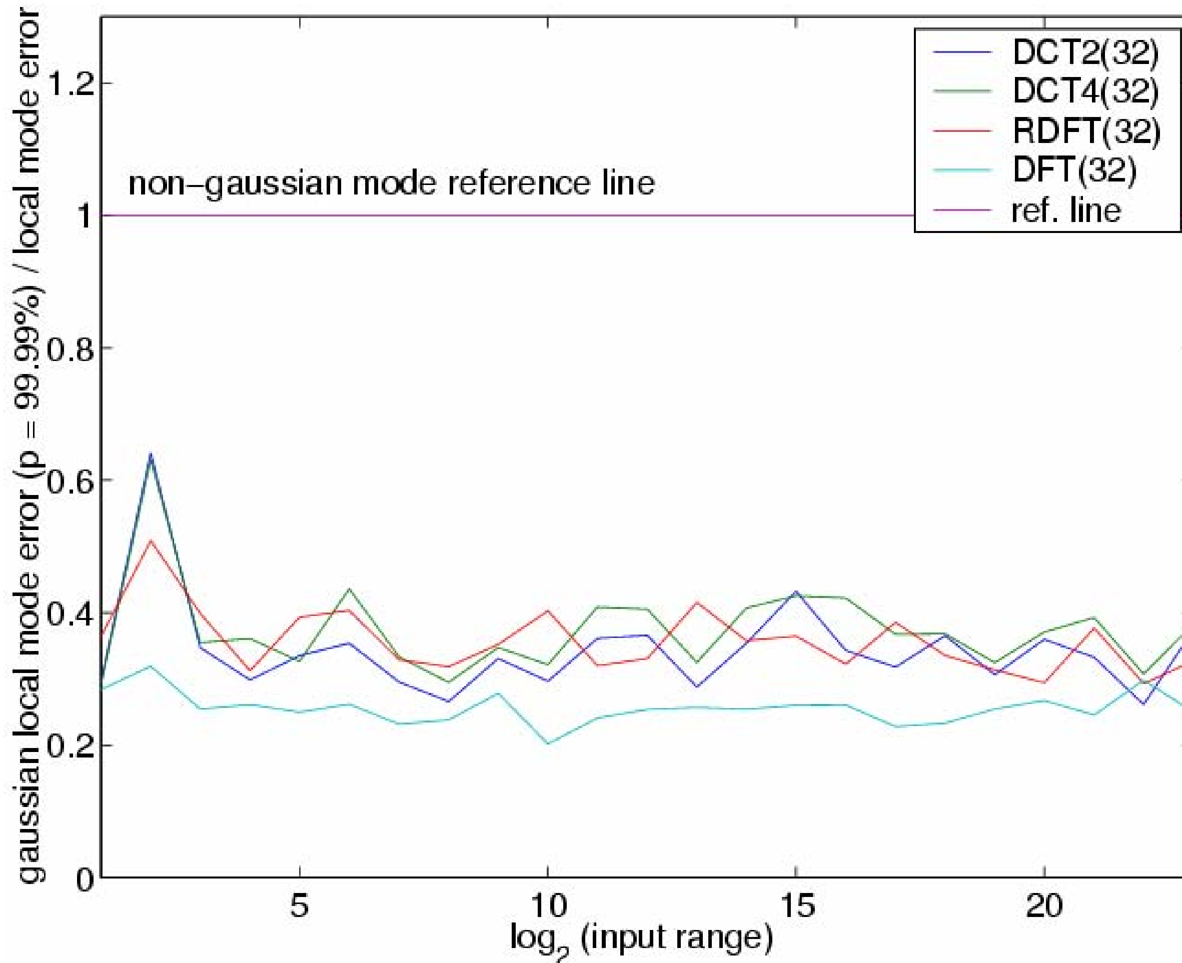
several transforms

local mode a factor of 1.5-2 better

Carnegie Mellon

SPIRAL

Local vs. Gaussian Local Mode



99.99% confidence for each variable

gain: about a factor of 2.5-4

Carnegie Mellon

SPIRAL

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:

    prob(overflow) < p

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

www.spiral.net