# Adaptive Mapping of Linear DSP Algorithms to Fixed-Point Arithmetic*

Lawrence J. Chang† Yevgen Voronenko† Markus Püschel†

## Introduction

Embedded DSP (digital signal processing) applications are typically implemented using fixed point arithmetic; in hardware to reduce area requirements and increase throughput, but also in software since most embedded processors do not provide floating point arithmetic. Consequently, the developer is confronted with the difficult task of deciding on the fixed point format, i.e., the number of integer and fractional bits to avoid overflow and ensure sufficient accuracy. For software implementations, the entire bitwidth is fixed, typically at 32, which means that increasing the representable range (number of integer bits) reduces the available accuracy (number of fractional bits) and vice-versa.

In this paper we present a compiler that translates a floating point C implementation of a linear DSP kernel, such as a discrete Fourier or wavelet transform, into a high accuracy fixed point C implementation. The inputs to the compiler are a floating point arithmetic C program and the range of the input vector elements. First, the compiler statically analyzes the program in a single pass using a recently developed tool that uses affine arithmetic modeling [1]. Then, in the *global mode*, the compiler determines the global fixed point format with the least number of integer bits (and thus the highest accuracy) that guarantees to avoid overflow and outputs the corresponding code. More interesting is the *local mode*, in which the compiler determines the best format *independently* for each variable, thus further pushing the possible accuracy. The compiler is currently limited to straightline code; an extension to loop code is in development.

Further, we used the SPIRAL code generator [2] to generate numerically robust implementations as input to our compiler, thus automating the entire design flow of creating high accuracy fixed point implementations for linear DSP kernels. Experiments with different transforms show that by choosing the formats independently (local mode) the accuracy can be improved by a factor of up to 5 in terms of a norm-based error measure.

## Adaptive Fixed-Point Mapping for High Accuracy

Our approach to generating a high accuracy fixed point implementation for a DSP transform $T$ consists of the following two high-level steps; the second step is our main contribution.

- We generate a numerically robust initial *floating point* implementation for $T$ using SPIRAL.
- We translate this implementation into a high accuracy fixed point implementation using the input range as additional information.

**Generating a Robust Initial Implementation.** SPIRAL is a generator for fast, platform-adapted implementations of DSP transforms and filters. SPIRAL operates in a feedback loop that generates, for a given transform, alternative algorithms and implementations to find the best match to the given platform. The feedback loop is driven by the measured runtimes of the generated codes; by replacing it with a norm-based accuracy measure, we use SPIRAL to generate numerically robust code.

**Adaptive Translation into Fixed Point Code.** To translate a floating point implementation into fixed point format, the crucial task is to determine the maximal range of each occurring variable. The tool in [1] uses affine arithmetic modeling to achieve this statically with a single pass through the code. The basic idea is to represent each variable $x$ by an *affine expression*

$$\hat{x} = x + \sum x_i \epsilon_i, \quad x_i > 0,$$

*This work was supported by NSF through awards ACR-0234293, SYS-0310941, and ITR/NGS-0325687.

†Dept. of Electrical and Computer Engineering, Carnegie Mellon University, {lchang,yvoronen,pueschel}@ece.cmu.edu .

# Report Documentation Page

*Form Approved*
*OMB No. 0704-0188*



| 1. REPORT DATE | 2. REPORT TYPE | 3. DATES COVERED |
|---|---|---|
| **01 FEB 2005** | **N/A** | **-** |

| 4. TITLE AND SUBTITLE | 5a. CONTRACT NUMBER |
|---|---|
| **Adaptive Mapping of Linear DSP Algorithms to Fixed-Point Arithmetic** | 5b. GRANT NUMBER |
| | 5c. PROGRAM ELEMENT NUMBER |
| 6. AUTHOR(S) | 5d. PROJECT NUMBER |
| | 5e. TASK NUMBER |
| | 5f. WORK UNIT NUMBER |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Dept. of Electrical and Computer Engineering,Carnegie Mellon University** | 8. PERFORMING ORGANIZATION REPORT NUMBER |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSOR/MONITOR'S ACRONYM(S) |
| | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) |

12. DISTRIBUTION/AVAILABILITY STATEMENT
**Approved for public release, distribution unlimited**

13. SUPPLEMENTARY NOTES
**See also ADM00001742, HPEC-7 Volume 1, Proceedings of the Eighth Annual High Performance Embedded Computing (HPEC) Workshops, 28-30 September 2004 Volume 1., The original document contains color images.**

14. ABSTRACT

15. SUBJECT TERMS

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT **unclassified** | b. ABSTRACT **unclassified** | c. THIS PAGE **unclassified** | **UU** | **24** | |

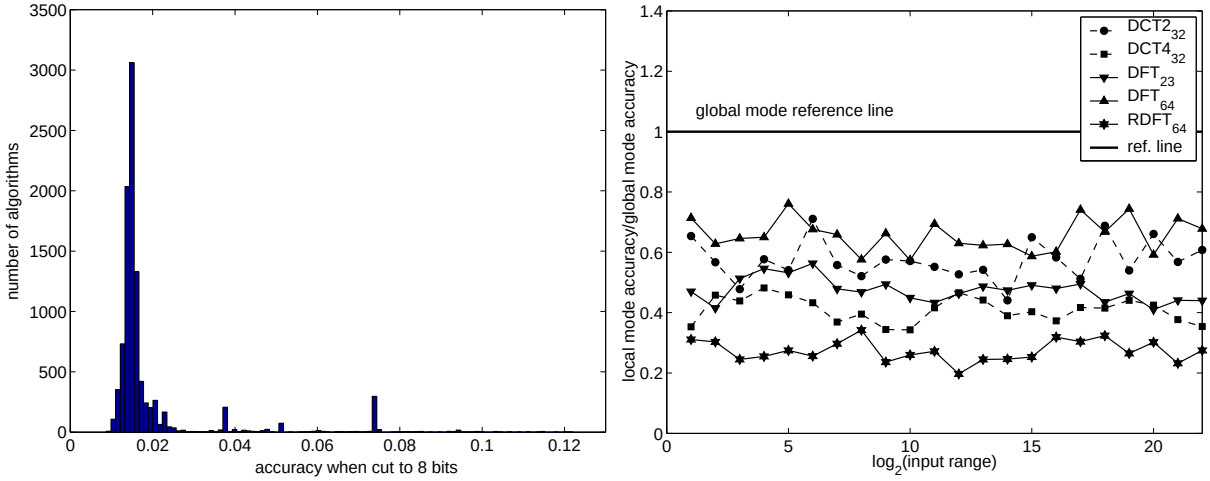



**Figure 1**: Left: accuracy histogram of 10,000 SPIRAL generated algorithms for a DCT, type 2, size 32. Right: relative accuracy of "local method" vs. "global method" (lower is better).

where the $\epsilon_i$ are random variables uniformly distributed in $[-1, 1]$. Intuitively, some of the $\epsilon_i$'s capture the range of each variable and others the uncertainty due to finite precision effects in the computation. Starting from the input, these affine expressions are computed for every variable in the code; each finite precision operation introduces a new error variable. For example, a global input range of $[-N, N]$ corresponds to affine expressions of the form $0 + N\epsilon_1$, i.e., at the input the entire uncertainty is due to range. For further details on the method see [1].

From the affine expression for a variable, its maximal range is obtained by setting all $\epsilon_i$ to 1 and -1. In the global mode, we determine the number of integer bits through the largest occurring range among all variables. In the local mode, the format of each variable is chosen independently.

## Results

Figure 1 (left) shows a robustness histogram of 10,000 SPIRAL generated algorithms for a DCT (type 2) of size 32. The robustness measure compares a floating point implementation to an 8-bit fixed-point implementation for each algorithm.[1] Most algorithms are within a factor of 2–3. Using SPIRAL's search mechanism we generate a robust algorithm as input to our compiler.

Figure 1 (right) shows the benefit of choosing independent fixed point formats (local mode) versus choosing a global format. Each line represents a transform; the $x$-axis shows the logarithm of the chosen input range (e.g., 10 means a range of $[-2^{10}, 2^{10}]$); the $y$-axis shows the relative accuracy of local vs. global. The best improvement of a factor of 3–5 is achieved for a real DFT (RDFT).

**Conclusion.** Our compiler achieves two main goals in the targeted domain. First, we free the developer from choosing a suitable fixed-point format by hand. Second, we obliterate the need for extensive simulations, since the generated code provably avoids overflow by construction. By using our compiler as backend to SPIRAL, the design flow is fully automated.

## References

[1] Claire F. Fang, Rob A. Rutenbar, and Tsuhan Chen. Fast, Accurate Static Analysis for Fixed-Point Finite Precision Effects in DSP Designs. In *Proc. ICCAD*, 2003.

[2] M. Püschel, B. Singer, J. Xiong, J. M. F. Moura, J. Johnson, D. Padua, M. Veloso, and R. W. Johnson. SPIRAL: A Generator for Platform-Adapted Libraries of Signal Processing Algorithms. *International Journal of High Performance Computing Applications*, 18(1):21–45, 2004. http://www.spiral.net.

[1] More precisely, we apply both implementations to all standard base vectors to create the (almost) exact transform matrix $M$ and the approximation $\tilde{M}$ and measure the matrix infinity norm $||M - \tilde{M}||_\infty$ of the difference.

# 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



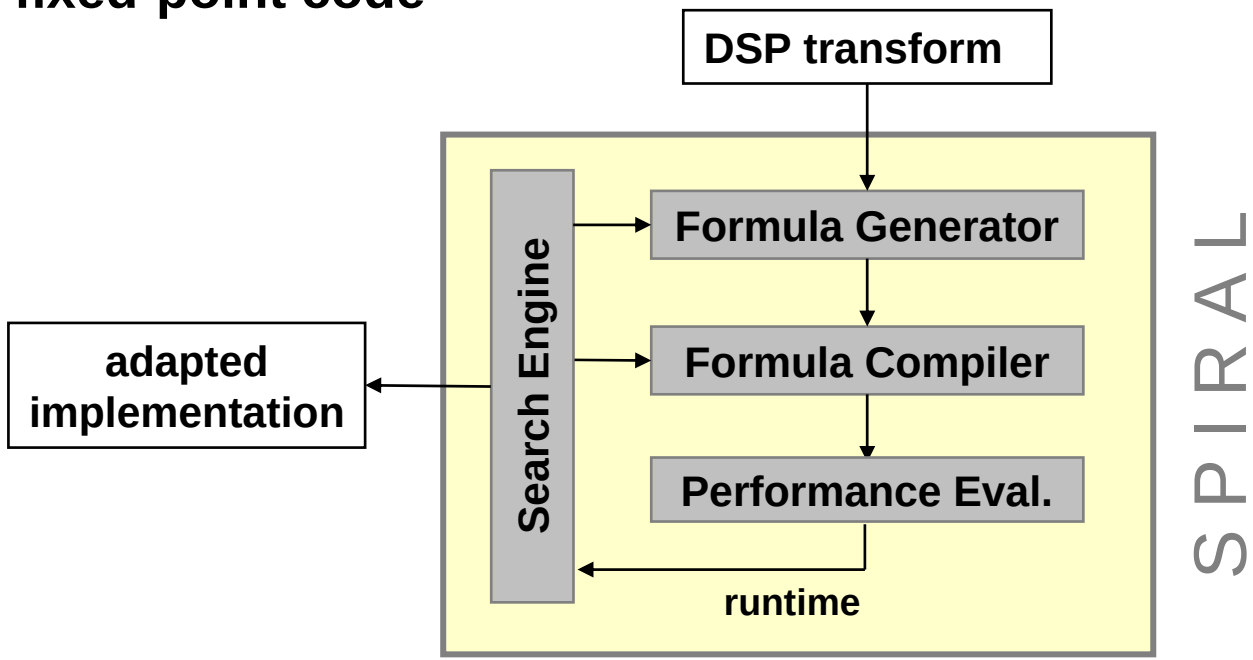
# Outline

- 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



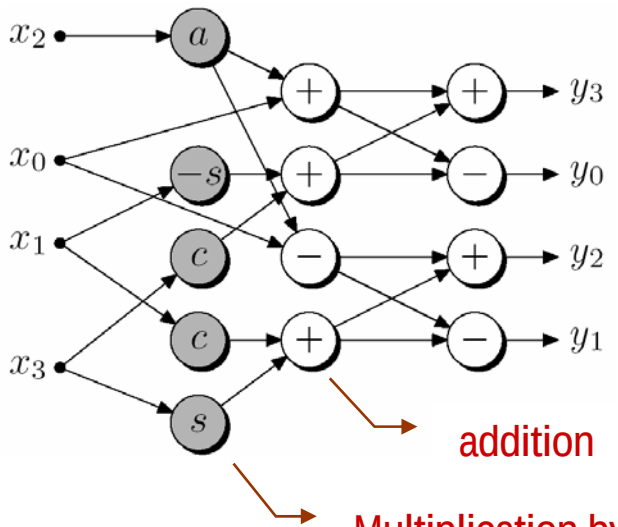

www.spiral.net

# 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

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

  - Data flow DAG (Directed Acyclic Graph)

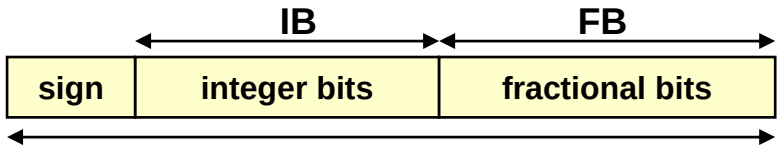



  - Program

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

# 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

| `a+b` | `a·b » fb` |
|---|---|
| addition | multiplication |

- Dynamic range:
  - $-2^{IB}$ ... $2^{IB}-1$
  - 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

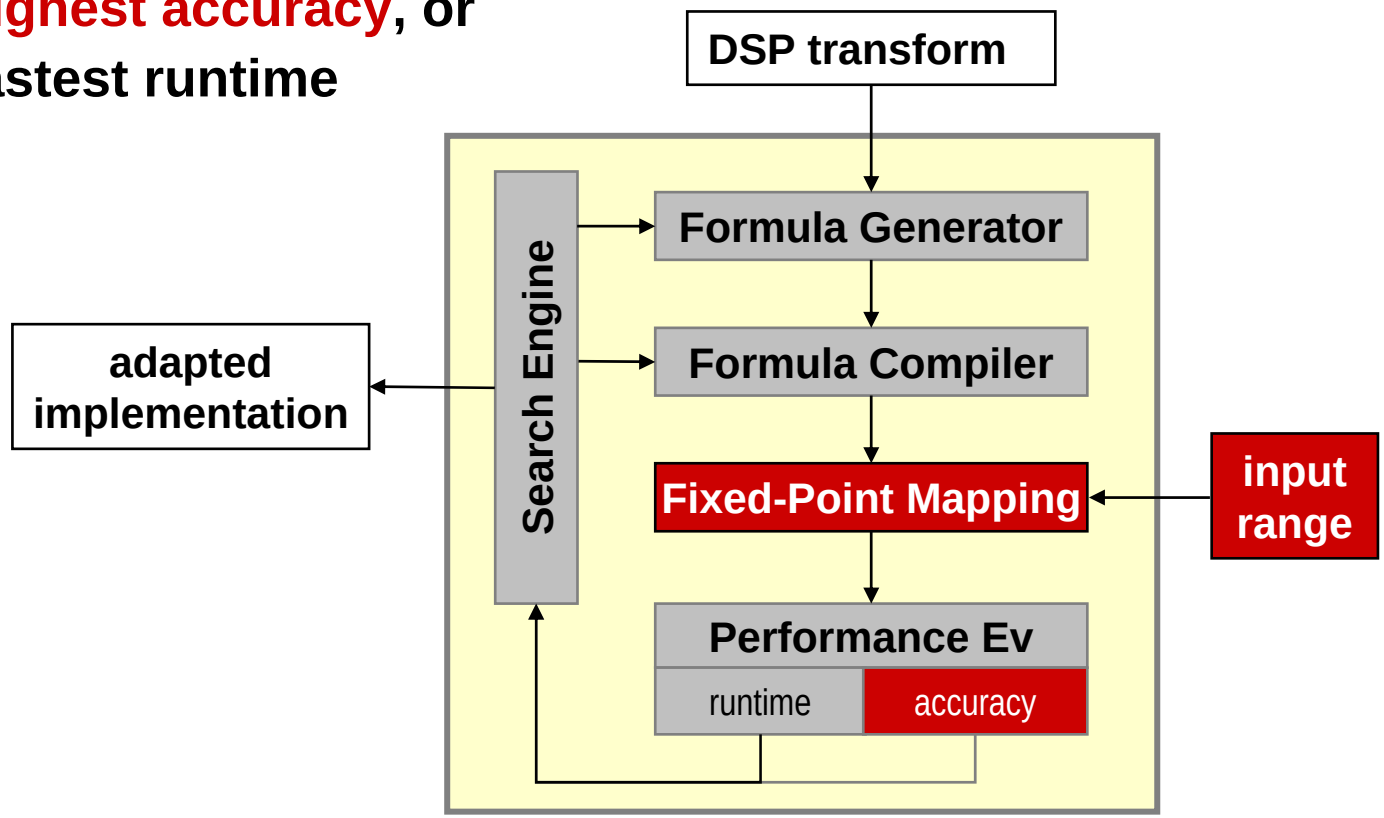
# Outline

- 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



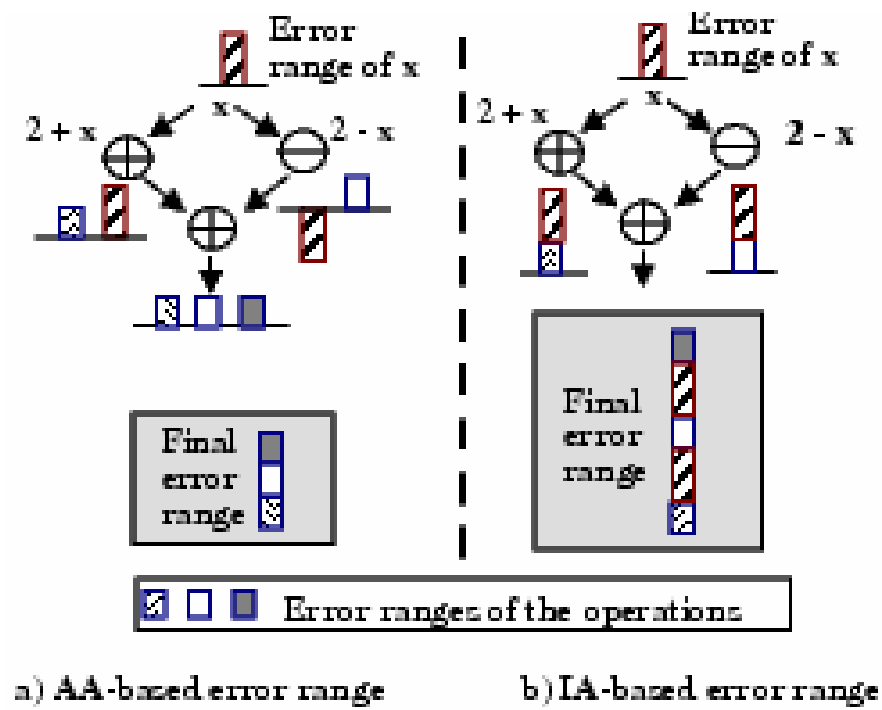
# Tool: Affine Arithmetic

- 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

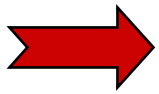
$$\mathcal{R}_{\min}(\mathcal{A}(v)) = \mathcal{R}_{\min}(\textstyle\sum_i c_i \cdot x_i) = \sum_i |c_i| \cdot \mathcal{R}_{\min}(x_i)$$
$$\mathcal{R}_{\max}(\mathcal{A}(v)) = \mathcal{R}_{\max}(\textstyle\sum_i c_i \cdot x_i) = \sum_i |c_i| \cdot \mathcal{R}_{\max}(x_i)$$



# Example

**Program**

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

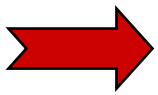


**Affine Expressions**

```
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 $\varepsilon_i$ 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 $A_\varepsilon$:

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

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**



# 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

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**

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

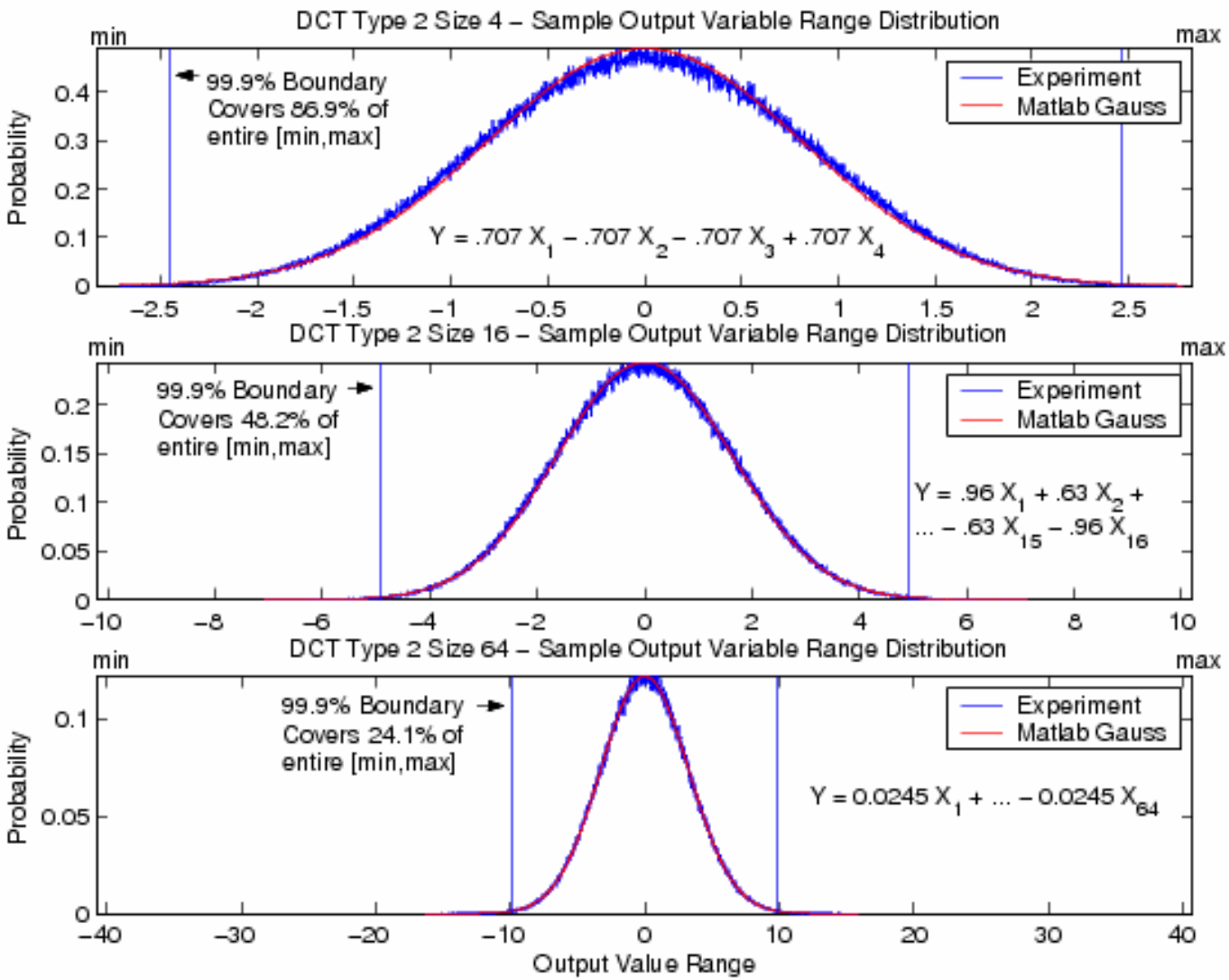
**However: not all values in [-M,M] are equally likely**

**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

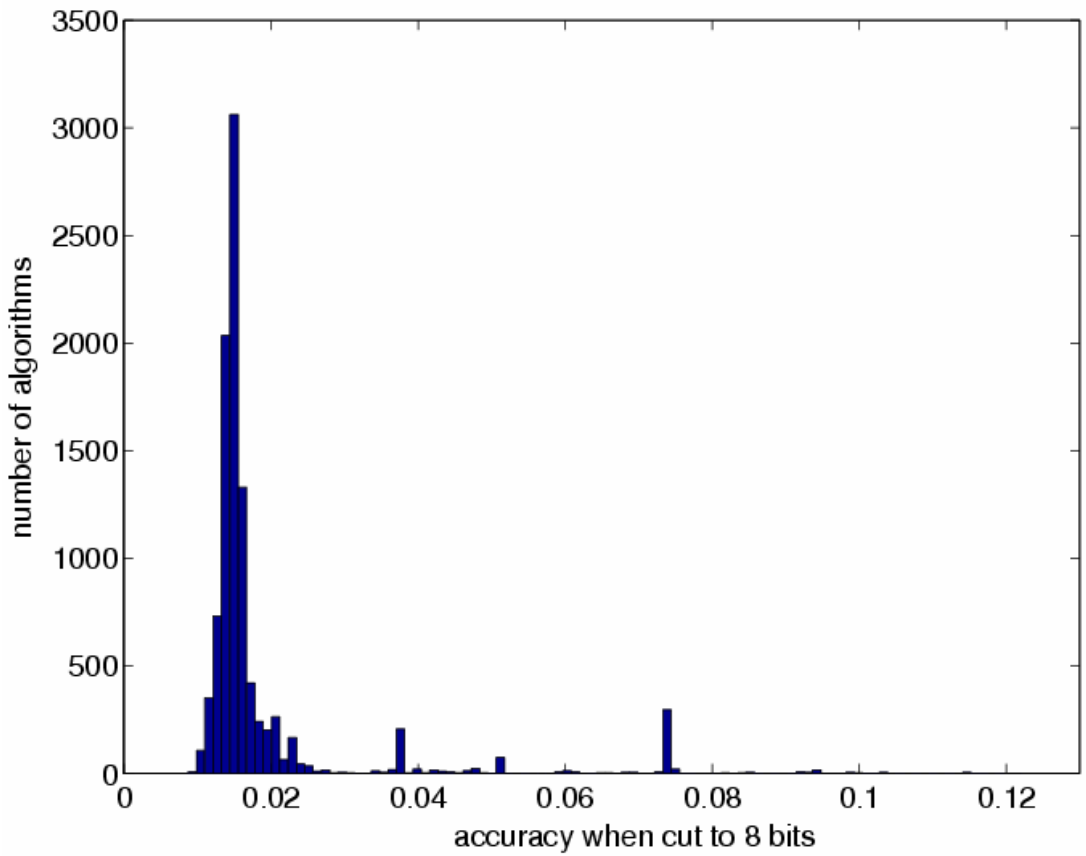
# Outline

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



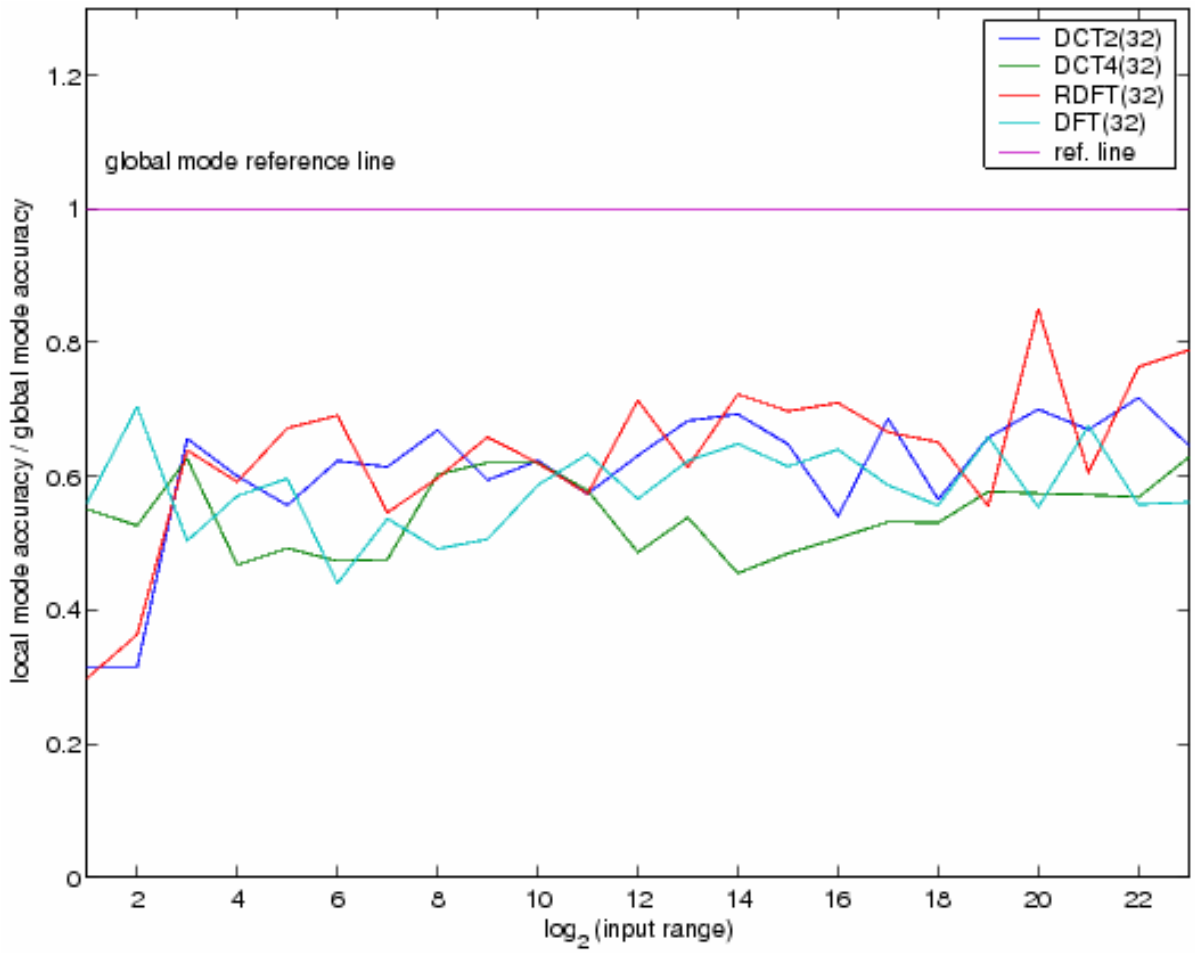
# Accuracy Histogram

**DCT, size 32**
**10,000 random algorithms**
**Spiral generated**




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

# Global vs. Local Mode



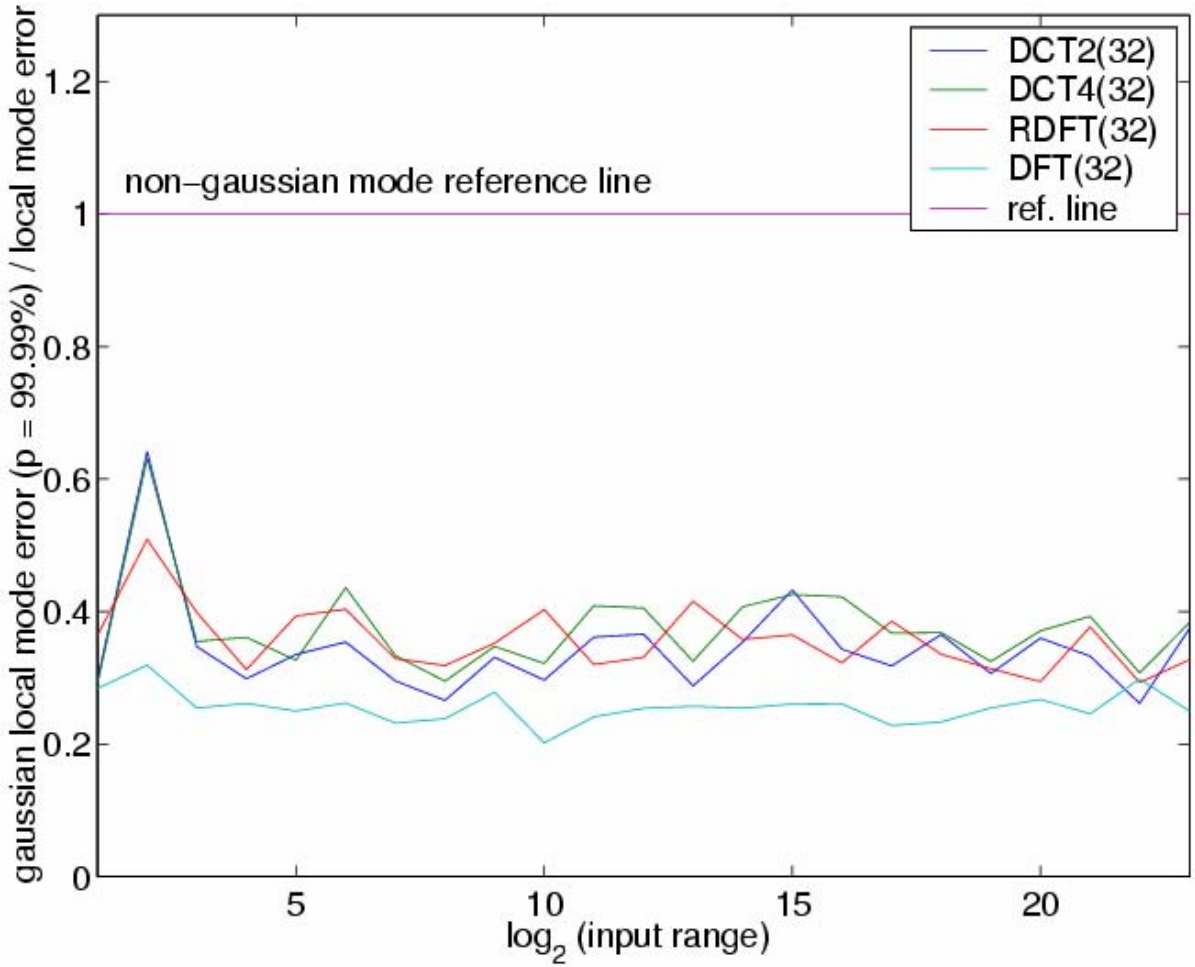

← several transforms

**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**