Toward Scalable Source Level Accuracy Analysis for Floating-point to Fixed-point Conversion

Gaël Deest, Tomofumi Yuki, Olivier Sentieys, Steven Derrien

Toward Scalable Source Level Accuracy Analysis for Floating-point to Fixed-point Conversion

Gaël Deest*, Tomofumi Yuki†, Olivier Sentieys*†, Steven Derrien*

*University of Rennes 1
†Inria
{first.last}@irisa.fr

***Abstract*—In embedded systems, many numerical algorithms are implemented with fixed-point arithmetic to meet area cost and power constraints. Fixed-point encoding decisions can significantly affect cost and performance. To evaluate their impact on accuracy, designers resort to simulations. Their high running-time prevents thorough exploration of the design-space. To address this issue, analytical modeling techniques have been proposed, but their applicability is limited by scalability issues. In this paper, we extend these techniques to a larger class of programs. We use polyhedral methods to extract a more compact, graph-based representation of the program. We validate our approach with a several image and signal processing algorithms.**

I. Introduction

Custom embedded hardware platforms require the programmer (designer) to address many challenges that do not appear in general purpose setting. One reason is that such implementations must satisfy tight design constraints such as performance, cost, energy, reliability, and so on. The importance of tools for efficient exploration of the design space is further emphasized by the market pressure towards shorter design cycles.

Most custom embedded hardware platforms are designed to provide high computing power for a low area/energy budget. Hardware support for floating-point operations is rarely provided, necessitating the use of fixed-point arithmetic to achieve performance. However, most digital signal and image processing specifications (in Matlab or C/C++) use floating-point. The implementation stage requires a costly (experiments [1] report up to 50% of design effort) floating-point to fixed-point conversion step, performed manually.

During the conversion process, designers try to derive implementations that maximize performance while retaining enough accuracy to preserve the functionality of the system. For most algorithms, the impact of encoding decisions on the overall system behavior is difficult to predict, and the performance/accuracy exploration therefore relies on simulations to evaluate encoding decisions. This simulation time is often a bottleneck, limiting the exploration of the solution space, leading to sub-optimal implementations.

To address this issue, analytical accuracy models have been proposed [2], [3], [4]. These methods focus on signal processing algorithms (e.g., FIR, IIR, FFT) that are modeled using Signal Flow Graphs (SFGs). These approaches fit well with model-based design flows (e.g., graphical dataflow languages) but are difficult to adapt in a compiler context, where the system model is implicitly specified as program source code.

Existing tools solve the problem by building an SFG model out of the program [3], [5], but are currently restricted to kernels operating on relatively small, one-dimensional inputs. The goal of this work is to extend the applicability of *automatic* analytical modeling techniques to a wider class of programs (in particular image processing algorithms). We focus on how to *systematically* build such models from source code in a more *efficient* and more *scalable* manner, rather than improving the quality (i.e., accuracy of the model) with respect to prior work.

The key idea in this work is the use of *convolutions* as a high-level operator to characterize input programs. Linear Time-Invariant (LTI) systems (and its generalization to higher dimensions) are convolutions by constants. Accuracy models for LTI systems rely on characterizing the system either with impulse response or transfer functions, which is significantly simplified by representing the program as combinations of convolutions. Specifically, our contributions are:

- a method to *automatically* build abstract models of the input program consisting of convolutions, employing a series of techniques from compilers,
- use of the abstract view of the program to model noise propagation, as well as the effect of quantization decisions, and
- prototype implementation and empirical validate our approach with several signal and image processing kernels.

The rest of this paper is organized as follows. In Section II, we present the necessary background on fixed-point accuracy analysis. We present the main limitations of existing approaches and highlight our contributions in Section III. Section IV introduces the Convolution System as a compact representation of the input program, which we use in Section V to build an analytical accuracy model of accuracy. We evaluate our approach with signal and image processing applications in Section VI. We discuss related work in Section VII, and conclude in Section VIII.

II. Background and Motivation

In this section, we introduce the problem of floating-point to fixed-point conversion, its impact on system accuracy, and

This work is co-funded by the European Union under the 7th Framework Programme under Grant Agreement ICT-287733.

the optimization problem it raises. We then give the necessary background on Linear-Shift Invariant (LSI) systems, the kind of systems targeted by our approach.

A. Floating-Point to Fixed-Point Conversion

In embedded systems, most multimedia and signal applications are implemented using fixed-point arithmetic instead of floating-point. In fixed-point arithmetic, real numbers are represented using scaled integers with a fixed number of bits used for the integer part and for the fractional part. We write $Q_{m,n}$ for the fixed-point format with m bits for the integer part (exclusive of the sign bit) and n bits for the fractional part. Such a format has $[-2^{-m}, 2^{-m} - 2^{-n}]$ as its dynamic range, with 2^{-n} as a quantization step.

When converting an initial program from a floating-point specification to a fixed-point implementation, one must choose a $Q_{m,n}$ format for every variable in the program. This choice must be done carefully as it can significantly alter the behavior of the system. Underestimating m may lead to overflows, i.e., operation output falls out of the dynamic range, which have high impact on system behavior. Underestimating n may lead to significant losses of precision due to quantization errors.

To handle these two types of errors, floating- to fixed-point tools operate in two stages:

1) Estimation of the dynamic range (one-time process).
2) Design space exploration (iterative process) that minimizes cost (energy, area) while satisfying execution time and accuracy constraints. The constraints on accuracy are expressed as Signal to Quantization Noise Ratio.

During the design space exploration, a large number of encoding combinations are considered before reaching a satisfying solution. Rapid evaluation of the effect of encoding decisions on system SQNR is therefore critical. For example, when targeting application-specific hardware, the designer has complete control over the operator and datapath wordlength. Using shorter fixed-point formats usually translates into both additional performance and lower area. The same holds (yet not as significantly) for embedded processors with sub-word SIMD execution units or non-uniform register sizes.

However, using shorter encodings impacts the overall system accuracy leading to a complex trade-off between accuracy (i.e., application quality degradation) cost, and performance, which is part of the design space exploration. During this exploration, designers try to obtain the best implementation while constraining the output noise to stay below some threshold. One category of approaches for evaluating the SQNR is based on simulations (e.g., [6], [7]). The program is simulated a number of times with different inputs and the outputs from these runs are combined to evaluate the system accuracy. The main drawback is obviously simulation time: evaluating the impact on accuracy for one set of encoding decisions requires the program to be executed many times, slowing down the entire design space exploration. Another set of approaches analytically model the noise power as a mathematical expression depending on data wordlength to avoid long simulation time [8], [3], [5]. These approaches are discussed in Section VII.

B. Error Propagation in Linear Shift-Invariant Systems

Our approach, like most others, relies critically on the system being Linear and Shift-Invariant (LSI). The main difference lies in our ability to efficiently handle complex multidimensional systems such as 2D recursive filters, while most approaches are, in effect, limited to one-dimensional Linear Time-Invariant systems.

LSI systems are a convenient framework to study error propagation. In such a system, signals and errors do not interfere and may be considered independently. Let T be a linear multidimensional system, $\mathbf{X}$ an input signal and $\widehat{\mathbf{X}} = \mathbf{X} + \varepsilon$ be a perturbated input. Then:

$$T(\widehat{\mathbf{X}}) - T(\mathbf{X}) = T(\varepsilon).$$

In other words, the propagated error is simply the output of the system when applied to the error itself.

A linear system is said Linear and Shift-Invariant if a shift at its input translates in a shift at its output. More formally:

$$\forall \vec{u}, \quad T(\tau_{\vec{u}}(\mathbf{X})) = \tau_{\vec{u}}(T(\mathbf{X}))$$

where $\tau_{\vec{u}}$ represents a translation by vector $\vec{u}$. LSI systems can be fully characterized by their *impulse response* $\mathbf{h}$, the response of the system to a unit impulse at the origin:

$$T(\mathbf{X}) = \mathbf{h} * \mathbf{X},$$

where $*$ denotes the convolution operation. This work relies on a compact representation of a system, based on convolutions, to efficiently estimate its impulse response. This response is then used to propagate the statistical properties of errors and compute output noise power.

III. Contributions

In this paper, we present a technique for *automatic* evaluation of accuracy by program source code analysis. The key distinction from prior work in this area is that our method operates on a higher level of abstraction—convolutions—by explicitly modeling them as part of the program representation. The use of higher level abstraction give the following benefits:

- Improved *scalability*. The program representation becomes much more compact making the entire process much more scalable. Specifically, there is no need to flatten the control flow (unrolling of loops, etc.), which greatly reduces the number of nodes in our internal representation.
- Broadened *applicability*. Convolutions allow us to extend the applicability beyond LTI systems to its generalization to higher dimensional linear systems. Combined with scalability improvement, our approach can handle image processing applications.

As a simple example, consider an implementation of block FIR filter:

```
for (n=0; n<NBLOCKS; n++)
  y[n] = a*x[n] + b*x[n-1] + c*x[n-2];
```

Existing approaches for automatic accuracy evaluation represent the program with a low-level graph-based representation where each node models an operation [3], [5]. In order to

construct such representation, the entire control structure in the source program must be flattened. For this example, the `for` loop must be completely unrolled, which requires that the number of blocks (`NBLOCKS`) is a known constant. The number of nodes in the graph corresponds to the number of operations in the program; `NBLOCKS` $\times 5$ for this example.

In our work, the above program is represented as a single one-dimensional convolution over x using $[a \quad b \quad c]$ as kernel coefficients. Such a high-level representation of the program is sufficient for our purposes, since the most important property of the system, i.e., impulse response, is captured.

The low-level representation quickly becomes infeasible for image processing applications as they have far more operations. For instance, Gaussian blur filters for $N \times M$ images have $O(NM)$ number of operations. However, it can be represented as a single two-dimensional convolution when modeled with convolutions. While many computations may not be "convolutions" from a strict, mathematical, point of view, any weighted sum can be viewed as convolutions. Perceiving them as convolutions enables us to analyze many signal/image processing applications in a unified manner. This is the key insight behind this work.

Note that we do not address the closely related problem of dynamic range estimation in this work. It is important to emphasize that the dynamic range needs to be determined only once and is not part of the design space exploration. Thus, its execution time is not as critical as accuracy evaluation, and methods based on profiling (e.g., Chen and Singh [9]) can be used. Nonetheless, estimation of dynamic range based on convolutions is an interesting direction of future work.

IV. Convolution Systems

In this section, we present how we model input programs as convolutions. We first use dependence analysis techniques from polyhedral compilation to represent programs as Systems of Recurrence Equations, and then identify convolutions from the equational view of the program.

A. Systems of Recurrence Equations

Systems of Recurrence Equations (SREs) is a formalism that has been extensively studied in the context of systolic array synthesis [10], [11].

Given a set of functions $a_0, a_1, a_2, ...$, indexed by integer vectors, a system of recurrence equations is a set of equations of the form:

$$a_m(z) = \text{expr}\left(a_{i1}\left(f_{i1}(z)\right), a_{i2}\left(f_{i2}(z)\right), ...\right)$$

that define the functions a_i. Functions f_{ij} map integer vectors to other integer vectors, and are restricted to affine functions. In addition, the functions a_x are associated with a validity domain, characterized by an integer polyhedron.

B. Extracting SREs from Loop Programs

The key link between SREs and programs is the Array Dataflow Analysis [12]. ADA is a powerful dependence analysis that gives exact dependence information. It captures *instance-wise* and *element-wise* dependence, and hence is said to be *exact*. Instance-wise means that each iteration of the loop, or each *instance* of an operation, is distinguished from the others. Element-wise means that which *element* of the array is being accessed are captured in the analysis.

As a simple example, consider the following:

```
for (i=0; i <N; i++)
S0: x[i] = 0;
    for (j = 0; j <M; j++)
S1:   x[i] += a[j];
```

Without instance-wise information, the analysis result only says `S1` depends on `S0` and `S1`. Without element-wise information, an instance of `S1` at $\langle i, j\rangle$ depends on all instances of `S0`, and `S1`, since they both write to the variable x.

With array dataflow analysis, the dependence information found for the read $x[i]$ in `S1` is the following:

$$\text{Producer of } x[i] \text{ at } S1\langle i,j\rangle = \begin{cases} j = 0 & : S0\langle i\rangle \\ j > 0 & : S1\langle i, j-1\rangle \end{cases}$$

The result of array dataflow analysis can also be viewed as a system of recurrence equations [12]. Intuitively, each statement can be considered to store its result in its own array, indexed by its instance vector. With exact dependence information, every read can be mapped to the appropriate element of this single assignment array, effectively constructing a functional specification. The above example can be re-written as the following SRE:

$$S0(i) = 0$$
$$S1(i,j) = \begin{cases} j = 0 & : S0(i) + a(j) \\ j > 0 & : S1(i, j-1) + a(j) \end{cases}$$

The domains of `S0` and `S1` are constructed from loop bounds. When the loop bounds are affine expressions, the domain of a statement is always a polyhedron. If statements with affine guards simply add additional constraints to the polyhedron. For example, `S0` in the following code:

```
for (i=0; i <N; i++)
  if (i>c) S0
```

has domain $\mathcal{D}_{S0} = \{i | 0 \le i < N \wedge i > c\} = \{i | c < i < N\}$.

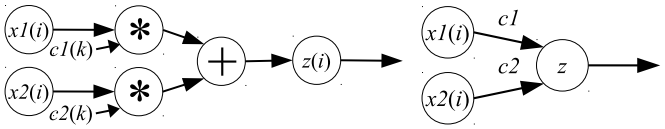
C. The Convolution Operator

We define the convolution operator $(*)$ as the following:

$$y = x * c$$
$$\equiv y(\vec{i}) = \sum_{\vec{k} \in \mathcal{D}_c} c(\vec{k}) x(\vec{i} + \vec{k})$$

where c and x are equations in the SRE that respectively define the convolution coefficients and the input dataset. For a d-dimensional convolution, the equations, c, x, and y are all defined over d-dimensional domain indexed by an integer vector $\vec{i}$. The domain of c, $\mathcal{D}_c$ defines the window where the convolution is applied, and c is expected to be constants. The equation x should refer to the input dataset or outputs from convolution, and hence transitively to input.

We call an SRE a *convolution system* if it consists entirely of convolutions, and its combinations by additions. Figure 1 illustrates its graph representation.



(a) Explicit (b) Compact

Fig. 1: Graph representation of convolution system $z(i) = (x1 * c1) + (x2 * c2)$. In the explicit representation, edges represent flow of data, and nodes represents *collections* of operations or data. Note that a single node represents more than one operation/data. In the compact representation, edges represent a convolution by the kernel associated with the edge, and the sum of the convolution results becomes the value of a node if it has multiple incoming edges. Indices of nodes are not shown except for input nodes in the compact representation.

D. Identifying Convolutions

Identifying convolutions is an instance of the algorithm recognition problem, which is known to be undecidable in general. Hence, we do not claim to have a general solution, but our approach can find many variations of convolutions found in signal/image processing applications.

Starting from an SRE, convolutions are identified by the following three steps:

1) Preprocessing.
2) Alignment of domains.
3) Extraction of kernel coefficients.

We illustrate the identification process with the Deriche filter in the following. Figure 2 shows fragments of code implementing the Deriche filter, and corresponding SREs.

Preprocessing: This step consists of inlining simple copy equations, distributing multiplications, and merging sequence of additions as a single n-ary addition (summation). Since the convolutions we target are those by constant coefficients, they can be viewed as weighted sums. This is why we first extract summations as a preprocessing step. The summations are then reorganized into those that share the same data variable.

Constant coefficients are either given directly as constants or as references to scalars and/or arrays. We assume that the names of constant coefficient scalars/arrays (such as `a1` through `a8` in Figure 2) are known.

Applying the preprocessing to the equations in Figure 2 gives the following:

$$\begin{aligned} S1(i,j) &= \mathtt{sum}(\mathtt{a1} \times \mathtt{in}[i][j], \mathtt{a2} \times \mathtt{in}[i][j-1]) \\ &\quad + \mathtt{sum}(\mathtt{b1} \times S1(i,j-1), \mathtt{b2} \times S1(i,j-2)) \qquad (1) \\ S5(i,j) &= S1(i,j) + y2(i,j) \\ S6(j,i) &= \mathtt{sum}(\mathtt{a7} \times S5(i+1,j), \mathtt{a8} \times S5(i+2,j)) \\ &\quad + \mathtt{sum}(\mathtt{b1} \times S6(i+1,j), \mathtt{b2} \times S6(i+2,j)) \\ S11(i,j) &= y1(i,j) + S6(j,i) \end{aligned}$$

where the summations are convolution candidates.

Alignment: The aim of the alignment step is to "normalize" the domains of equations in the SRE. In this implementation of the Deriche filter, the horizontal passes visit columns of the image in the inner loop, where the vertical passes visit rows of the image (after horizontal filter) in the inner loop. This can be viewed as applying horizontal pass on the *transpose* of the image. In fact, the code for horizontal and vertical passes are almost identical due to this transposed view of the image. Thus, it is important to align the domains of equations in the SRE, or the "view" of the image, to appropriately identify the kernel coefficients, including its orientation, of the convolutions.

The alignment problem in our context is closely related to that in the context of uniformization/localization of dependences [13], [14]). Uniformization is a process of transforming the equations such that all accesses in the definition of the equations are within some constant distance. The definitions of $S6$ and $S11$ does not satisfy this property (e.g., $S11(i,j)$ uses $S6(j,i)$, which is a sign of transposition). We use an adaptation of the alignment algorithms used for uniformization to align the equations. We obtain the following after alignment for $S6$ and $S11$ in our running example:

$$\begin{aligned} S6(i,j) &= \mathtt{sum}(\mathtt{a7} \times S5(i+1,j), \mathtt{a8} \times S5(i+2,j)) \\ &\quad + \mathtt{sum}(\mathtt{b1} \times S6(i+1,j), \mathtt{b2} \times S6(i+2,j)) \\ S11(i,j) &= y1(i,j) + S6(i,j) \end{aligned}$$

Note that all references in the right hand side of the equations are i, j plus some constant offset. For the purpose of convolution detection, it is sufficient to have all dependences share the same linear part, and not necessarily uniform.

Kernel Extraction: Once the equations are aligned, we are ready to extract the kernel coefficients that characterize the convolution. The domain of the kernel, i.e., window of the convolution, is defined by the access functions to the data variables. As an example, a summation $y(n) = \mathtt{sum}(\mathtt{a} \times x(n), \mathtt{b} \times x(n-2), \mathtt{c} \times x(n-3))$, has a window of length $4; -3 \le n \le 0$, where the coefficients are $[\mathtt{c} \quad \mathtt{b} \quad 0 \quad \mathtt{a}]$.

The equations for $S1$ and $S6$ are identified as a combination of four convolutions:

$$\begin{aligned} S1(i,j) &= (\mathtt{in} * c1) + (S1 * c2) \\ S6(i,j) &= (S5 * c3) + (S6 * c4) \end{aligned}$$

where the coefficients $c1$ through $c4$ are as follows:

$$\begin{aligned} c1(i,j) &= [\mathtt{a1} \quad \mathtt{a2}], \mathcal{D}_{c1} = \{i,j | -1 \le i \le 0 \wedge j = 0\} \\ c2(i,j) &= [\mathtt{b1} \quad \mathtt{b2}], \mathcal{D}_{c2} = \{i,j | -2 \le i \le -1 \wedge j = 0\} \\ c3(i,j) &= \begin{bmatrix} \mathtt{a8} \\ \mathtt{a7} \end{bmatrix}, \mathcal{D}_{c3} = \{i,j | i = 0 \wedge 1 \le j \le 2\} \\ c4(i,j) &= \begin{bmatrix} \mathtt{b2} \\ \mathtt{b1} \end{bmatrix}, \mathcal{D}_{c4} = \{i,j | i = 0 \wedge 1 \le j \le 2\} \end{aligned}$$

Note that the domain of convolution is two-dimensional, even though it is effectively one-dimensional. This is necessary to distinguish horizontal and vertical passes, and the domains of coefficients always have the same dimensionality as the data being convolved in our representation of convolutions. The constructed convolution systems have corresponding graph representations, as illustrated in Figure 3.

```
//horizontal pass (right)
for (i=0; i<W; i++) {
   ym1=ym2=xm1=0;
   for (j=0; j<H; j++) {
S1:  y1[i][j] = a1*in[i][j] + a2*xm1
                + b1*ym1         + b2*ym2;
S2:  xm1 = in[i][j];
S3:  ym2 = ym1;
S4:  ym1 = y1[i][j];
   }
}

//left pass not shown for brevity
y2[i][j] = ...

for (i=0; i<W; i++)
   for (j=0; j<H; j++)
S5:  t[i][j] = y1[i][j] + y2[i][j];
```

(a) Code for horizontal passes

```
//vertical pass (down) not shown for brevity
y1[i][j] = ...

//vertical pass (up)
for (j=0; j<H; j++) {
   tp1=tp2=yp1=yp2=0;
   for (i=W-1; i>=0; i--) {
S6:  y2[i][j] = a7*tp1 + a8*tp2 + b1*yp1 + b2*yp2;
S7:  tp2 = tp1;
S8:  tp1 = t[i][j];
S9:  yp2 = yp1;
S10: yp1 = y2[i][j];
   }
}
for (i=0; i<W; i++)
   for (j=0; j<H; j++)
S11: T[i][j] = y1[i][j] + y2[i][j];
```

(b) Code for vertical passes

$$
\begin{aligned}
S1(i,j) &= \mathtt{a1} \times \mathtt{in}[i][j] + \mathtt{a2} \times S2(i,j-1) \\
&\quad + \mathtt{b1} \times S4(i,j-1) + \mathtt{b2} \times S3(i,j-1) \\
S2(i,j) &= \mathtt{in}[i][j] \\
S3(i,j) &= S4(i,j-1) \\
S4(i,j) &= S1(i,j) \\
S5(i,j) &= S1(i,j) + y2(i,j)
\end{aligned}
$$

(c) SRE corresponding to horizontal passes

$$
\begin{aligned}
S6(j,i) &= \mathtt{a7} \times S8(j,i+1) + \mathtt{a8} \times S7(j,i+1) \\
&\quad + \mathtt{b1} \times S10(j,i+1) + \mathtt{b2} \times S9(j,i+1) \\
S7(j,i) &= S8(j,i+1) \\
S8(j,i) &= S5(i,j) \\
S9(j,i) &= S10(j,i+1) \\
S10(j,i) &= S6(j,i) \\
S11(i,j) &= y1(i,j) + S6(j,i)
\end{aligned}
$$

(d) SRE corresponding to vertical passes

Fig. 2: Code fragments from Deriche filter and corresponding SREs. Deriche filter applies total of 4 IIR filters in different directions (right, left, up, down). The SREs shown exclude boundary cases (e.g., when $j = 0$) to simplify our presentation.

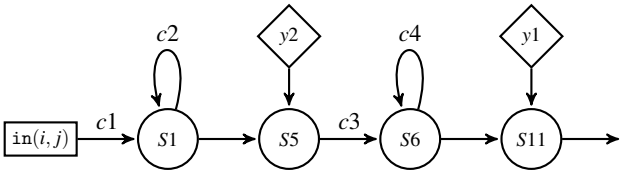


Fig. 3: Graph representation of the convolution system for Deriche fragment using compact representation. Unlabeled edges can be viewed as "convolutions" by identity coefficient. The nodes $y1$ and $y2$ would be identical to nodes $S1$ and $S6$ if the full code were analyzed.

E. Scope of the Analysis

The powerful dependence analysis is made possible by focusing on a subset of programs where the control flow is static. Informally, the scope of the analysis is defined to be loop nests where all array accesses, loop bounds, and `if` guards are (quasi-)affine expressions of the surrounding loop indices. Quasi-affine expressions are affine expressions with modulo by constants, which allow us to express conditionals such as: `if (i%2==0)`. Moreover, the analysis handles sequences of imperfectly nested loops.

In addition, it is not necessarily for the full application to satisfy the above restriction. For instance, house keeping code that does not affect the main computation can be excluded from the analysis. Such code regions can be identified by less accurate but more general dependence analysis techniques (e.g., simple name-based dependences, and/or dependence vectors).

Recall that the boundary conditions have been ignored in this section. Although it is possible to construct convolution systems with all the boundary cases, an approximation of the system without the boundaries are used in the subsequent sections as well. The main branches of SREs can usually be identified by inspecting the conditions of each branch, since boundary conditions have less "effective" dimensions due to equalities. By constructing systems that consists of these branches, the resulting system becomes an approximation the underlying system that focus on the steady state.

V. Accuracy Model Construction

This section describes how the convolution system introduced in the previous section is transformed to an accuracy model of the program. This model, parameterized by quantization decisions, is well suited to automatic wordlength optimization. In this section, we use a subset of the convolution system developed in Section IV to illustrate our approach. The sub-graph corresponding to the first horizontal pass is reconstructed in Figure 4.

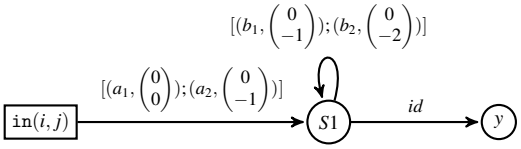


Fig. 4: Convolution system of the horizontal stage (right) of the Deriche filter. Edges are labeled with a compact representation of the convolution kernel. This graph corresponds to the computation $y(i,j) = a_1 in(i,j) + a_2 in(i,j-1) + b_1 y(i,j-1) + b_1 y(i,j-2)$ in Equation 1.

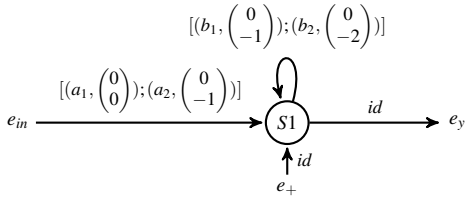


Fig. 5: Transformed system with noise sources. Input is replaced by e_{in}, the quantization error of `in`. Node e_y represents the sum of quantization errors in the computation of y.

A. Adding Noise Sources

The first step is to modify the system to model the computation at the quantization error level. This process is illustrated in Figure 5. Inputs are replaced by Quantization Noise Sources (QNSs) to model quantizations of inputs. A QNS is also added to every internal and output node to model the errors in the computation represented by this node. Note that these noise sources may correspond to several quantizations, as quantizations may occur after every arithmetic operation.

B. Modeling Contributions of Noise Sources to Outputs

In a linear system, the contribution of each noise source to an output may be considered independently. For every pair of noise source s and output o, a sub-system (sub-graph) of the system is constructed to independently model the contribution of s to o. The set of sub-systems modeling the contributions to o is denoted as Err_o.

These subsystems are defined as follows. Let G be the graph representing the contribution of a noise source s to o. A node v belongs to G if there is a path from s to v and from v to o. An edge $v_1 \rightarrow v_2$ belongs to G if $v_1, v_2 \in G$. This process is illustrated in Figure 6.

C. Propagating Noise Characteristics

To estimate output accuracy, the statistical properties of the noises are propagated to the outputs. Let μ_s and σ_s^2 be, respectively, the mean and variance of the noise of some noise source s. Instances of this noise are supposed independent and identically distributed. Let $\mathbf{h}_{s,o}$ be the (possibly multi-dimensional) impulse response of the system from s to o. The mean and variance of the propagated noise are:

$$\mu_{s,o} = \mu_s \sum_{\vec{v}} \mathbf{h}_{s,o}(\vec{v}) \qquad \sigma^2_{s,o} = \sigma^2_s \sum_{\vec{v}} \mathbf{h}^2_{s,o}(\vec{v}), \tag{2}$$

where vector $\vec{v}$ spans the space of coordinates.

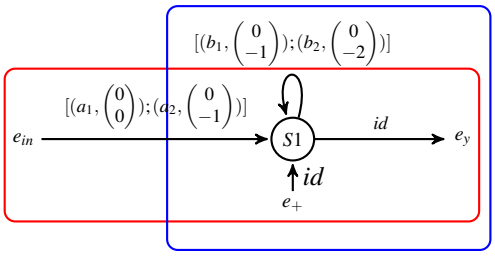


Fig. 6: Decomposition of the horizontal Deriche filter graph. Each rounded box defines a subgraph which captures the propagation of a noise source to output y.

To enable rapid computation of $\mu_{s,o}$ and $\sigma^2_{s,o}$, the above sums must be pre-computed by evaluating the impulse response on a large window around the origin. By definition, the impulse response is the response of the system to a unit impulse. We can thus obtain it by executing the system (SRE) with a unit impulse as input data-set, and then use the corresponding output as the system impulse response (this can be seen as a form of abstract simulation).

Since this impulse response can be infinite, some approximations may be required. If the system is numerically stable, it is assumed that the impulse response reaches zero beyond some reasonable range. We use this fact to bound the size of the impulse response obtained through our method.

If the graph does not contain any cycle, its impulse response is finite and can be computed exactly. The same is true for the parts of the graph that are not reachable from any cycle, such the as the first convolution in the Deriche filter in Figure 4. This observation can be used to further optimize the computation. Finally, this process can be easily parallelized.

D. Computing the Noise Power Function

Noise mean and variance at output o are given by summing those of individual noise contributions:

$$\mu_o = \sum_{s \in Err_o} \mu_{s,o} \qquad \sigma^2_o = \sum_{s \in Err_o} \sigma^2_{s,o} \tag{3}$$

Noise power is given by:

$$P_o = \mu_o^2 + \sigma_o^2 \tag{4}$$

E. Applying the Model

To compute noise power for a given fixed-point specification, (μ_s, σ_s^2) must be determined for every noise source s in the graph. It can then be computed with equations 2, 3 and 4.

As we observed in Section V-A, a noise source can actually correspond to several quantizations. Consider the Equation 1 from Deriche example (Figure 4). Quantizations may occur after every operation in the expression and at the end of the computation, when the result is constrained to the format of its final destination. In our example, the total error is thus the sum of six potential quantization errors.

TABLE I: Model construction time for our tool and ID.fix.

Algorithm	ID.Fix (s)	Our tool (s)
IIR8	23.1	20.5
Sobel (32×32)	169.1	9.2
Sobel (64×64)	2173.1	9.7
Sobel (128×128)	-	9.4
Gaussian blur (32×32)	160.1	10.2
Gaussian blur (64×64)	2010.9	9.5
Gaussian blur (128×128)	-	9.4
Deriche blur (16×16)	-	6.5

TABLE II: Validation of our model against simulations.

Algorithm	Simulation (dB)	Our tool (dB)	Error (dB/%)
IIR8	-17.80	-17.84	-0.04/-0.2%
Sobel	11.62	12.04	0.42/3.6%
Gauss	3.78	3.78	<0.01/0.1%
Deriche	-18.01	-18.06	-0.05/-2.78%

The properties (mean, variance) of these intermediate noises can be determined from the quantization mode (truncation, rounding, convergent rounding), the precision of the operands and the quantization step. As these noises are independent, μ_s and σ_s^2 can be computed by summation as in Equation 3.

VI. Experimental Validation

The proposed analysis has been implemented within a compiler framework, and validated with signal and image processing applications. We contrast our scalability to another accuracy analysis tool based on the work by Ménard et al. [3], called ID.fix[1]. We used a Linux machine with Intel core i7 (3.0GHz) and 8GB of memory for our experiments.

Table I shows the time required to construct the accuracy model for our benchmark applications. Although ID.Fix is not designed to handle 2D algorithms, it can still process them by seeing each pixel as a different input. The difference in scalability can be clearly observed with image processing applications. Our approach is unaffected by the input image size whereas ID.fix is unable to handle the Deriche filter on 16×16 images. 2D recursive filters, such as Deriche, have wide impulse response, and 16×16 image would not be sufficient to approximate the error.

Once the model is constructed, evaluating the model for a given quantization decision takes less than a second in all cases. The models constructed by ID.fix also take less than a second to evaluate. Table II demonstrates the accuracy of the constructed models validated against simulations.

VII. Discussion and Related work

Previous attempts to perform accuracy analysis from program sources have significant limitations that restrict their use to small kernels. Most tools [4], [15] ignore the program control-flow and remain at the basic-block level (i.e., DFG level). Hence, they fail at capturing inter basic-block interactions and it is unclear how such tools can handle sequence of loop nests accessing same arrays with non-trivial indexing functions, which are common in image/video processing.

The work by Ménard et al. [3] addresses this problem by completely flattening the program control-flow to obtain a single (large) SFG for the whole program, leading to scalability issues as pointed out in Section III. In contrast, our approach is able to capture more complex propagation patterns (inter-block, instance-wise) without resorting to flattening/unrolling.

A. Accuracy Analysis from a Compiler Point of View

From a strict accuracy modeling perspective, our approach may appear as a straightforward extension of earlier work on LTI systems to multi-dimensional signal processing systems (LSI). Indeed, we do use the same type of domain specific knowledge, which is understanding that the underlying system is LSI. Such knowledge can directly be exposed in the system specifications when appropriate model-based design approaches are used (e.g., Simulink), but are not available when working with arbitrary source programs.

Our work addresses two fundamental issues, (i) how to infer such domain specific knowledge from input source programs (i.e., from a compiler point of view), and (ii) how to represent the system in a compact way such that the analysis scales to programs beyond simple signal processing kernels. In this respect, our contribution also has some connections with algorithm recognition problems [16] and its scope goes beyond accuracy modeling techniques (at least as defined by the signal processing community).

One may argue that all these domain specific knowledge can be expressed in the source program as annotations/pragmas. However, it is extremely difficult to define a reasonably small set of annotations that can capture the domain specific knowledge, when the tool is expected to handle many different ways of implementing an algorithm. Our observation is that trying to specify sufficient information through annotations ends up asking the programmer to specify the from system à la model-based design, eliminating the benefits of automation. Moreover, the tool is incapable of verifying if the given annotations are "correct" if it relies all important information to be supplied by the user.

B. Extensions to non-LTI Systems

As already pointed out by previous authors, the limitation to LSI systems prevents from using the technique when non linear operators are involved in the algorithm or when the system impulse response changes over time, as is the case for adaptive filters. However, several extensions to the noise propagation model have been proposed to handle such situations. For example, Constantinides [17] proposed to handle non-LTI, non-recursive filters by linearizing noise contributions. A variety of methods was developed to compute the linearization coefficients [3], [4], [18]. Recently, Rocher et al. showed how to handle any recursive and non-recursive filter composed of smooth operations [5].

Most of these extensions follow the same principle: they

[1] http://idfix.gforge.inria.fr/

use additional profiling information[2] (statistic over signal values, or cross-correlation information) to built a LTI/LSI approximation of the real system. This approximation is then used to derive the error noise statistics. Since they ultimately recast the problem into a LTI/LSI formulation, we believe that such extensions can be easily integrated within our approach.

C. Working with non-polyhedral Programs

As explained in Section IV, our approach operates on programs with regular control structure that can be represented using the polyhedral model. Many/most image signal processing algorithms fall into this category, and those that do not are unlikely to correspond to LTI systems anyway. However, there exist a few exceptions, the most notable one being the Fast Fourier Transform. In such a case, the best solution to the accuracy analysis problem is probably to rely on a domain/application specific noise propagation model that would capture the recursive nature of the algorithm (borrowing from the philosophy of tools such as Spiral [19]).

D. Arithmetic Accuracy Bounds in Numerical Computation

The problem of numerical accuracy analysis in program is a widely studied topic. Most work in the field aim at deriving safe error bounds between finite and infinite precision algorithms (or between two finite precision specifications) [20], or to help finding the cause of numerical inaccuracies [21]. Boland and Constantinides [22] showed how scalability issues could be addressed in that context by trading the quality of bounds for faster accuracy evaluation. The problem we address here is very different as we are interested in the statistical properties (such as mean and power) of the noise error rather than its bounds. Our work could be extended to derive bounds on error instead of statistical properties.

VIII. Conclusion

In this work, we have proposed a scalable approach to the problem of accuracy modeling for floating-point to fixed-point conversion. Our approach works by inferring high-level convolution operations from the original source code, and explicitly modeling them as part of the program representation. The two main outcomes of the technique are (i) improved scalability with respect to prior work and (ii) the ability to handle higher dimensional linear systems. Future work directions include extending the approach to handle correlated quantification noises, and to support non-linear, but differentiable, operations.

References

[1] M. Clark, M. M., D. Jackson, and D. Linebarger, "Accelerating Fixed-Point Design for MB-OFDM UWB Systems," *CommsDesign*, January 2005.

[2] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "The Multiple Wordlength Paradigm," in *Proceedings of the 9th IEEE Symposium on Field-Programmable Custom Computing Machines*, 2001, pp. 51–60.

[3] D. Ménard, R. Rocher, and O. Sentieys, "Analytical Fixed-Point Accuracy Evaluation in Linear Time-Invariant Systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 55, no. 10, pp. 3197–3208, 2008.

[2]This profiling stage is performed only once on the initial floating point data-set, it should not be confused with simulation based approaches.

[4] G. Caffarena, C. Carreras, J. A. López, and A. F. Herrero, "SQNR Estimation of Fixed-Point DSP Algorithms," *EURASIP Journal on Advances in Signal Processing*, pp. 21:1–21:12, 2010.

[5] R. Rocher, D. Ménard, P. Scalart, and O. Sentieys, "Analytical Approach for Numerical Accuracy Estimation of Fixed-Point Systems Based on Smooth Operations," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 10, pp. 2326–2339, 2012.

[6] P. Belanovic and M. Rupp, "Automated Floating-Point to Fixed-point Conversion with the fixify Environment," in *Proceedings of the 16th IEEE International Workshop on Rapid System Prototyping*, 2005, pp. 172–178.

[7] S. Kim and W. Sung, "Finite wordlength effects analysis and wordlength optimization of a multiplier-adder based 8× 8 2D-IDCT architecture," in *Proceedings of the 1996 IEEE International Symposium on Circuits and Systems*, vol. 2, 1996, pp. 672–675.

[8] G. A. Constantinides, P. Y. Cheung, and W. Luk, "Wordlength optimization for linear digital signal processing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 10, pp. 1432–1442, 2003.

[9] D. Chen and D. P. Singh, "Profile-Guided Floating- to Fixed-point Conversion for Hybrid FPGA-processor Applications," *ACM Transactions on Architecture and Code Optimization*, vol. 9, no. 4, pp. 43:1–43:25, 2013.

[10] P. Quinton, "Automatic Synthesis of Systolic Arrays from Uniform Recurrent Equations," in *Proceedings of the 11th Annual International Symposium on Computer Architecture*, 1984, pp. 208–214.

[11] S. Rajopadhye, S. Purushothaman, and R. Fujimoto, "On Synthesizing Systolic Arrays from Recurrence Equations with Linear Dependencies," in *Proceedings of the 6th Conference on Foundations of Software Technology and Theoretical Computer Science*, 1986, pp. 488–503.

[12] P. Feautrier, "Dataflow Analysis of Array and Scalar references," *International Journal of Parallel Programming*, vol. 20, no. 1, pp. 23–53, 1991.

[13] Z. Chen and W. Shang, "On Uniformization of Affine Dependence Algorithms," in *Proceedings of the 4th IEEE Symposium on Parallel and Distributed Processing*, 1992, pp. 128–137.

[14] V. Van Dongen and P. Quinton, "Uniformization of Linear Recurrence Equations: a Step Toward the Automatic Synthesis of Systolic Arrays," in *Proceedings of the International Conference on Systolic Arrays*, 1988, pp. 473–482.

[15] D. Novo, S. El Alaoui, and P. Ienne, "Accuracy vs Speed Tradeoffs in the Estimation of Fixed-point Errors on Linear Time-invariant Systems," in *Proceedings of the 16th Conference on Design, Automation and Test in Europe*, 2013, pp. 15–20.

[16] C. Alias and D. Barthou, "On the Recognition of Algorithm Templates," *Electr. Notes Theor. Comput. Sci.*, vol. 82, no. 2, pp. 395–409, 2003.

[17] G. A. Constantinides, "Perturbation Analysis for Word-length Optimization," in *Proceedings of the 11th IEEE Symposium on Field-Programmable Custom Computing Machines*, 2003, pp. 81–90.

[18] C. Shi and R. W. Brodersen, "A Perturbation Theory on Statistical Quantization Effects in Fixed-Point DSP with Non-Stationary Input," in *Proceedings of the 2004 International Symposium on Circuits and Systems*, 2004, pp. 373–376.

[19] R. Koutsoyannis, P. Milder, C. Berger, M. Glick, J. Hoe, and M. Puschel, "Improving fixed-point accuracy of FFT cores in O-OFDM systems," in *Proceedings of the 2004 International Symposium on Acoustics, Speech and Signal Processing*, March 2012, pp. 1585–1588.

[20] E. Goubault and S. Putot, "Static Analysis of Finite Precision Computations," in *Proceedings of the 12th International Conference on Verification, Model Checking, and Abstract Interpretation*, 2011, pp. 232–247.

[21] F. Benz, A. Hildebrandt, and S. Hack, "A Dynamic Program Analysis to Find Floating-point Accuracy Problems," in *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2012, pp. 453–462.

[22] D. Boland and G. A. Constantinides, "A scalable approach for automated precision analysis," in *Proceedings of the 20th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2012, pp. 185–194.