Custom-Optimized Multiplierless Implementations of DSP Algorithms

Markus Püschel, Adam C. Zelinski, and James C. Hoe
Electrical and Computer Engineering Department
Carnegie Mellon University
Pittsburgh, PA, U.S.A.
{pueschel, acz, jhoe}@ece.cmu.edu

Abstract

Linear DSP kernels such as transforms and filters are comprised exclusively of additions and multiplications by constants. These multiplications may be realized as networks of additions and wired shifts in hardware. The cost of such a "multiplierless" implementation is determined by the number of additions, which in turn depends on the value and precision of these constants. For a given transform or filter, the set of constants and their required precision is affected by algorithmic and implementation choices and hence provides a degree of freedom for optimization. In this paper we present an automated method to generate, for a given linear transform, a minimum addition multiplierless implementation that satisfies a given quality constraint. The method combines automatic algorithm selection to improve numerical robustness and automatic search methods to minimize constant precisions in a chosen algorithm. We present experiments that show the trade-offs between cost and quality, including custom optimizations of the transforms used in JPEG image and MP3 audio decoders.

Keywords

DSP, transforms, multiplierless, finite precision, fixed point, optimization

INTRODUCTION

Designers of digital signal processing (DSP) applications use various techniques to reduce area and power requirements for mobile and embedded applications. One important design point is the *finite-precision, multiplierless* implementation of linear DSP kernels. Most commonly used DSP kernels, such as transforms and filters, are linear; they consist exclusively of additions and multiplications by constants. In a multiplierless implementation, the constant multiplications are realized as networks of additions and wired shifts. For example, the multiplication $y = 5x$ would be realized as $y = (x \ll 2)+x$, i.e., by one addition and one shift. Multiplierless implementations are primarily of interest for hardware, but are also useful in software, e.g., in fixed point processors that have a large multiplication overhead.

A typical design flow for creating and optimizing a finite-precision, multiplierless implementation of a transform is shown in Figure 1. First, the designer has to choose a numerically robust algorithm to allow for the least possible precision in the final fixed point implementation. Algorithm selection is a *manual* process that requires the designer to understand relevant DSP concepts and literature. Given that there is often a large variety of possible algorithms (each with a complex structure), this step often requires significant effort. The second step is the accuracy configuration. The goal is to reduce the precision of the multiplicative constants (and thus the number of additions) without exceeding a given error constraint. The two major problems are 1) the exponentially large number of different configurations of constant precisions and 2) the fact that reducing the precision of one or several constants has a virtually unpredictable impact on the output error and is strongly dependent on the chosen error measure. The chosen error measure, in turn, depends on the application context. Note that the precision of the datapath and the precision of the constant multiplications can be chosen independently (e.g., the multiplication $y = 5x$ requires one addition independent of the bitwidth of x, y). The result of the accuracy configuration is a data flow graph consisting of additions, subtractions, and shifts. At this point the input/output function of the final implementation is determined. In the final synthesis step, the actual implementation is produced after undergoing additional optimization (e.g., scheduling and register allocation for software, and resource allocation and scheduling for hardware).

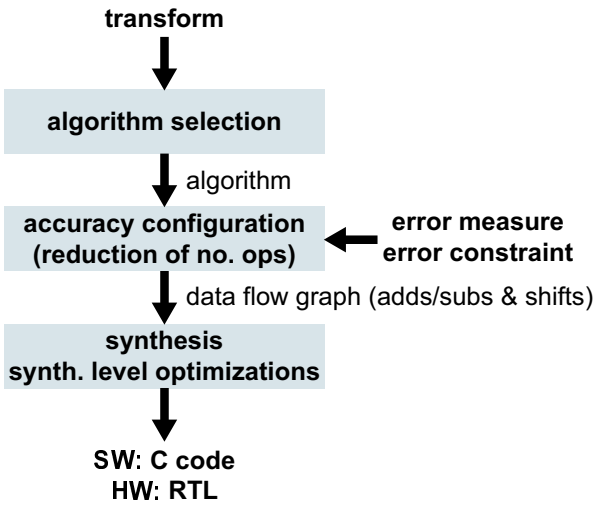

Figure 1. Design flow for creating and optimizing a multiplierless implementation of a DSP transform.

Automatic constraint-based cost reduction. In this paper we present, for the domain of linear DSP transforms, an

approach for automating the first two steps in Figure 1: algorithm selection and accuracy configuration. Given a transform T, an arbitrary error measure E and an error threshold $E_{\max}$, our method consists of the following two high-level steps:

- We automatically select and generate a numerically robust algorithm for T.
- We use intelligent search methods to find, for a given datapath bitwidth, the lowest cost (i.e., least number of additions) configuration of constant precisions in this algorithm such that the corresponding approximate transform $\widetilde{T}$ satisfies $E(\widetilde{T}) \leq E_{\max}$. Further, we use addition chains to optimally expand constant multiplications into additions and shifts.

To the best of our knowledge, we claim novelty in three directions by 1) providing the first method for automatic robust algorithm selection; 2) addressing *arbitrary*, including application driven, error measures; and 3) providing the first automatic method for independently configuring constant precisions across a large set of transforms.

Our approach complements existing research and tools that address the third (bottom) block in Figure 1, thus offering the possibility of an entirely automated design flow from the selected transform to the final implementation.

Related work. The main emphasis of current high-level synthesis research addresses the bottom block in Figure 1, the synthesis and optimization of a data flow graph to a final implementation (refer to [1]). The synthesis and optimization described in this paper are at a higher level, in the mathematic and algorithmic domains.

To date, research on multiplierless implementations of linear DSP kernels has been focused on filters; very little effort has been devoted to the more complex domain of transforms, which are the subject of this paper. Exceptions include [12, 3]. Reference [12] presents an in-depth study of multiplierless implementations of a specific algorithm of the DCT, type II, size 8, used in JPEG image compression; [3] expands on this work by introducing the idea of a greedy search to find the best constant precisions. This work, however, requires the knowledge of a robust algorithm as a starting point, considers only the above DCT, and uses a method to convert multiplications by constants into additions and shifts that is, in general, suboptimal. We presented a first generalization in [17] by considering different transforms and introducing different types of feedback-driven searches. In this paper we expand this work into a fully automated design flow by automating the algorithm selection, defining proper robustness metrics, and presenting an evaluation of the cost-accuracy tradeoffs in various application scenarios.

Our work is also related to research on mapping floating point into fixed point implementations. For example, [4] provides a tool to analyze the error propagation in a data flow graph to determine the required range and a bound for the error margin of the computation, but the precision is determined only *globally* and the method is not easily adaptable for non-numerical error measures. Other recent papers have also addressed issues in fixed point representation and quantization errors (e.g., [15], [16]).

Organization of this paper. The Background section provides the necessary background on DSP transforms, algorithms, and the techniques for mapping constant multiplications into additions and shifts. The next section explains automatic Algorithm Selection (the first block in Figure 1); the section after that explains automatic Accuracy Configuration (the second block in Figure 1). The final section presents experimental results of custom-optimized multiplierless implementations of transforms within several application contexts. One particular experiment reduces the number of additions required by the inverse modified discrete cosine transform (IMDCT) within MP3 audio decoding to 288 additions while still maintaining "limited accuracy" as defined by the MP3 International Organization for Standardization (ISO) standard.

BACKGROUND

In this section we provide the necessary background on DSP transforms, DSP transform algorithms, the SPIRAL code generator, and multiplierless implementation techniques.

DSP transforms, algorithms, and SPIRAL

Transforms. Mathematically, a (linear) DSP transform is a multiplication $y = Mx$, where x is an input vector (e.g., a sampled signal), M the transform matrix, and y the output vector (e.g., a transformed signal). Many different transforms are used in signal processing, including the discrete Fourier transform, the discrete cosine transforms (DCTs), and the discrete wavelet transform. In this paper, we consider three variants of DCTs that are important kernels in the JPEG and MPEG image/video/audio coding standards: type II, type IV, and the IMDCT. They are defined by the matrices

$$\begin{aligned}
\mathrm{DCT}_n^{(\mathrm{II})} &= \big[\cos \frac{k(2\ell+1)\pi}{2n}\big]_{0\leq k,\ell<n},\\
\mathrm{DCT}_n^{(\mathrm{IV})} &= \big[\cos \frac{(2k+1)(2\ell+1)\pi}{4n}\big]_{0\leq k,\ell<n},\\
\mathrm{IMDCT}_n &= \big[\cos \frac{(2k+1)(2\ell+1+n)}{4n}\big]_{0\leq k<2n,0\leq \ell<n}.
\end{aligned}$$

Note that the IMDCT_n is a $2n \times n$ matrix.

Transform algorithms. For each transform of size n there is a surprisingly large number of different fast algorithms, which have similar, close to minimal arithmetic cost (typically of the order $O(n \log(n))$), but have different structures and different numerical accuracies. The reason for this variety lies in the recursive structure of the algorithms. For a given transform, there are various ways of computing it using other, smaller transforms. The combination of these choices leads to a combinatorial explosion: the number of algorithms grows exponentially with n. For example, for a $\mathrm{DCT}^{(\mathrm{II})}$, SPIRAL (introduced in the next paragraph) reports 82 different algorithms for size 8, but as many as 1639236012 algorithms for size 32.

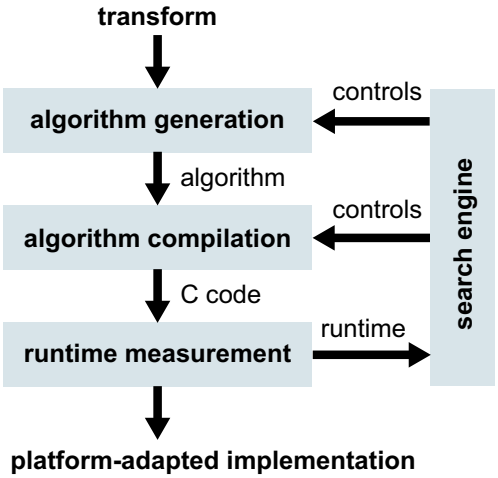

Figure 2. Architecture of SPIRAL.

SPIRAL is a generator for optimized software implementations of DSP transforms [14]; SPIRAL's architecture is shown in Figure 2. For a given transform, SPIRAL uses a rule mechanism to generate fast algorithms, represented in a mathematical language called *Signal Processing Language* (SPL). The algorithms are compiled into C code and their measured runtime is fed back into a search engine that triggers the generation of different algorithms with different implementation options (e.g., the degree of loop unrolling). Rather than using a brute force exploration of the entire algorithmic space, SPIRAL employs a sophisticated feedback-driven search engine to heuristically guide algorithm exploration. While this method does not guarantee that the optimal implementation will be uncovered, it finds close-to-optimal algorithms quickly and automatically. In practice, SPIRAL-generated competes with and, in some cases, even outperforms hand-tuned code (e.g., [5]).

Other forms of algorithm selection have been used to enable optimizations of hardware datapaths for performance [13]. For the purpose of this paper, we borrow the SPIRAL framework to generate numerically robust algorithms (the first block in Figure 1). This is accomplished by replacing SPIRAL's default runtime optimization metric with a robustness measure defined in the Algorithm Selection section.

Addition chains

Linear DSP transform algorithms compute a transform using exclusively additions and multiplications by constants. When multiplying by constants in hardware, costly full multipliers may be avoided by replacing them with networks of additions and shifts. Since shifts are realized in hardware by wires, we model the cost of such a multiplierless implementation as the number of additions. Now we will explain the *addition chain* method for optimally expanding a multiplication by a constant into the least number of additions, which offers a significant savings over the more commonly used canonical signed digit (CSD) method.

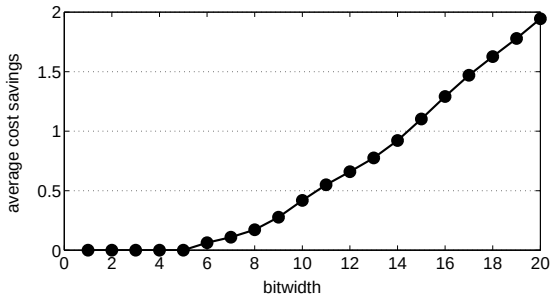

Figure 3. Average cost savings of addition chains versus CSD.

When approximating a constant c as a fixed point number, it takes the form

$$c \approx k/2^n,$$

where n is the number of fractional bits. Since the number of additions required to multiply by c is not affected by the position n of the decimal point, we restrict our discussion to integer fixed point numbers $c = k$. Given an integer k and its bit representation

$$k = \sum_{0 \le i < n} b_i 2^i, \quad b_i \in \{0, 1\}, \tag{1}$$

a direct method for multiplying by k would require as many additions as the number of non-zero b_i minus one, which can be as large as $n - 1$.

The well-known canonical signed digit (CSD) representation [10, chap. 6] expresses k as in (1), but allows $b_i \in \{\overline{1}, 0, 1\}$, where $\overline{1}$ stands for -1. The most salient feature of CSD is that it replaces each sequence of s 1's by a 1, followed by $s - 1$ 0's, followed by a $\overline{1}$. Furthermore, no two consecutive bits in a CSD representation are nonzero. The worst case scenario now requires $\lfloor (n-1)/2 \rfloor$ additions or subtractions. For example, $k = 315$ has the bit representation 100111011 and the CSD representation $101000\overline{1}0\overline{1}$.

It is known that CSD in general does not yield the minimum cost solution for multiplying by a constant k. The optimal method requires *addition chains* (or, more precisely, addition/subtraction/shift chains), which reuse intermediate results more flexibly. For example, the number $k = 1197$ has the CSD representation $1010\overline{1}0\overline{1}0\overline{1}01$, which requires 5 additions/subtractions. The best addition chain computes $y = kx$ using only 3 additions:

$$\begin{aligned} t &= x \ll 3 + x \\ y &= t \ll 7 + t \ll 2 + t \end{aligned}$$

For a given k, finding the optimal addition chain is known to be NP-complete [2]. Algorithmic methods exist to generate the solutions for numbers up to at least 20 bits, e.g., [6], which is the method we use. A maximum of five additions is needed by constants of up to 19 bits. Figure 3 shows the average savings (in additions/subtractions) achievable when using addition chains versus CSD for bitwidths up to 20.

ALGORITHM SELECTION

In this section, we explain how to automate the first block in Figure 1, the selection of a numerically robust algorithm for a given transform. As mentioned in the Background section, a given transform has many different algorithms. When implemented in fixed point arithmetic, these algorithms will exhibit different degrees of output quality degradation. In this section, we first define a measure of robustness and then explain how a numerically robust algorithm can be automatically selected by using SPIRAL (introduced in the Background section).

Robustness. Assume a given algorithm T exists, which represents the exact transform matrix M, i.e., $T = M$. When implemented in k-bit fixed point arithmetic, this algorithm represents an approximation of the matrix M, i.e., $\widetilde{T}_{k\text{-bit}} \approx M$. Thus, as a measure of robustness of the algorithm T, we use

$$N_k(T) = ||T - \widetilde{T}_{k\text{-bit}}||, \tag{2}$$

where $||\cdot||$ is a matrix norm. A good choice is a matrix norm that is *subordinate* to a vector norm (see [7] for more details on norms).

For a vector $x = (x_1, \ldots, x_n)$, the possible norm functions are jointly written as $||x||_p$, where $p \geq 1$ or $p = \infty$. Examples include the standard Euclidean norm ($p = 2$) and the maximum norm ($p = \infty$), defined respectively by

$$||x||_2 = \big(\sum_i |x_i|^2\big)^{1/2}, \quad ||x||_\infty = \max_i\{|x_i|\}.$$

For each vector norm $||\cdot||_p$ there is a unique matrix norm, the *subordinate norm*, also written as $||\cdot||_p$, that satisfies

$$||Mx||_p \leq ||M||_p \cdot ||x||_p, \quad \text{for all } M,\ x. \tag{3}$$

In particular, the matrix norm $||M||_2$ is the *spectral norm*, i.e., the largest singular value of M. In this paper, we will use $||M||_\infty$, the *max-row-sum norm*, since it is faster to compute:

$$||M||_\infty = \max_i\big\{\sum_j |M_{i,j}|\big\}.$$

Now we may motivate the use of the proposed robustness measure N_k in (2): namely, the measure itself is *input independent*, but may still be used to derive input dependent bounds. Assuming an input x and setting $y = Tx$ (the exact result) and $\tilde{y} = \widetilde{T}_{k\text{-bit}}x$ (the approximate result), and using (3), we derive the output error bound

$$||y - \tilde{y}||_\infty \leq ||T - \widetilde{T}_{k\text{-bit}}||_\infty ||x||_\infty = N_k(T)||x||_\infty.$$

For example, if we assume a given algorithm T and that x is an image of 8-bit color values lexicographically arranged into a vector, then $||x||_\infty \leq 255$, yielding the concrete error bound $255N_k(T)$.

Choosing a less robust algorithm results in a less accurate implementation under a given choice of fixed point format and limits the opportunity to reduce the precision of the fixed point computation. Figure 4 shows a histogram of $N_8(T)$ for all 82 algorithms for the $\mathrm{DCT}_8^{(\mathrm{II})}$, all 226 algorithms for the $\mathrm{DCT}_8^{(\mathrm{IV})}$, and all 44 algorithms for the IMDCT_{18}. (We chose different scales since the transforms have different numbers of multiplications and thus different accuracies.) The spread in robustness is large, showing the importance of *deliberately* choosing the right algorithm.

Using SPIRAL to generate robust algorithms. For this paper, we use SPIRAL's search mechanism to "find" a numerically robust algorithm. This is achieved by replacing the runtime measurement of an algorithm (see Figure 2) by the robustness measure N_k. This automates the first block in Figure 1.

ACCURACY CONFIGURATION

Once a numerically robust algorithm has been selected using SPIRAL, the remaining problem considered in this paper is how to choose the bitwidth of each multiplicative constant to minimize the number of additions without exceeding a given error constraint (the second block in Figure 1). Note that the bitwidth of each constant may be chosen independently, which yields an exponentially large space of alternatives that, in general, may not be exhaustively searched over.

The optimization problem. We formally state the optimization problem that we solve. Let T be the algorithm selected for the transform to be implemented. An approximation (finite-precision version) of T is denoted by $\widetilde{T}$. The cost of $\widetilde{T}$, $\mathrm{cost}(\widetilde{T})$, is the number of additions it requires to compute. Further, let an error measure E be given, which is a function that assigns each $\widetilde{T}$ a real value $E(\widetilde{T}) \geq 0$. We assume that a lower value of $E(\widetilde{T})$ is better. (The case of measures where a higher value is better is handled analogously.) Further, a quality threshold E_{max} is given. The problem can now be stated as follows.

> **Given:** an algorithm T, an error measure E, and an error threshold E_{max};
> **Find:** an approximation $\widetilde{T}$ with $E(\widetilde{T}) \leq E_{\text{max}}$ that minimizes $\mathrm{cost}(\widetilde{T})$.

We propose to solve this optimization problem through an automated intelligent search in the space of alternatives, i.e., in the space of different constant approximations. In this section, we first discuss different possible error measures E and then we investigate the size of the actual search space. Finally, we introduce the different search methods used for optimization.

Error measures. Our approach does not make any assumptions on the error measure E used. The only requirement is a reasonably efficient (fast) evaluation of $E(\widetilde{T})$, since the search methods introduced below will generate and evaluate many different $\widetilde{T}$ during the optimization process.

We consider two types of error measures E in this paper: 1) numerical error measures and 2) application-based error measures.

A *numerical error measure* computes the norm of the difference between the exact algorithm T and its approximation $\widetilde{T}$ as

$$E(T) = ||T - \widetilde{T}||, \tag{4}$$

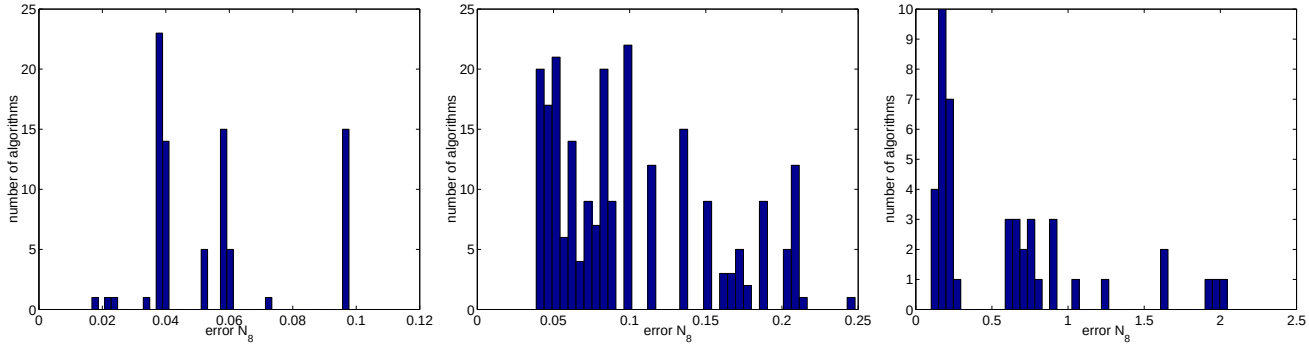

Figure 4. Robustness (N_8) histograms of algorithms generated by SPIRAL. From left to right: $\mathbf{DCT}_8^{(II)}$, $\mathbf{DCT}_8^{(IV)}$, $\mathbf{IMDCT}_{18}$.

similar to (2) (where the constant precisions were globally reduced). As explained in the Algorithm Selection section, we prefer a subordinate matrix norm in (4).

An *application-based error measure* evaluates $\widetilde{T}$ in the context of an application. Evaluation requires inserting code for $\widetilde{T}$ into a simulation or software implementation of the application. In the experiments in this paper we choose the second option. The code generation for $\widetilde{T}$ is provided by SPIRAL. For applications, we consider a JPEG decoder [8] containing a $\mathrm{DCT}_8^{(II)}$ and an MP3 audio decoder [11] containing an IMDCT_{18} and a $\mathrm{DCT}_{32}^{(II)}$. Both applications are implemented in software and use 32-bit fixed point arithmetic.

As a reasonable compliance test for the JPEG decoder, we use the peak signal-to-noise ratio (PSNR) of a compressed and then decompressed reference image compared to the original uncompressed image. (The compliance test defined by the JPEG standard is not commonly used.) A PSNR > 30 dB is considered reasonable for many applications. Note that for evaluation we only insert the approximation $\widetilde{T}$ into the decompression stage (not into the compression stage), because in many real-world scenarios we have no control over the compression process. Also, note that higher PSNR is better, i.e., it is not an error measure but rather a quality measure.

The MP3 audio decoding standard defines two levels of compliance [9]: limited accuracy (LA) and full compliance (FC). Both provide thresholds for the root-mean-square (RMS) difference between a certain reference audio file and the compressed-decompressed version. In addition, FC provides a threshold for the maximum difference. Similar to the JPEG experiment, we only place our approximated transforms $\widetilde{T}$ into the MP3 decoding stage.

Search space. Assume a given transform algorithm requires N multiplications by constants and the maximum datapath bitwidth is n. Then there are $(n+1)^N$ different bitwidth configurations. However, not all of them are meaningful. When reducing the bitwidth of a constant by one bit, the constant may not change, or, if it changes, it may still require the same number of additions. Clearly, among different bitwidths with the same complexity, only the largest (i.e., the most precise) has to be considered. For example, consider the constant $\cos(\pi/16)$ represented with a maximal bitwidth of 16 (all fractional bits). Among the 17 possible approximations (obtained by rounding), only 8 lead to distinct numbers, which are given in Table 1 together with their complexity; the ones considered in the search are marked with an "x". Note that a coarser (i.e., less precise) approximation may have a higher complexity.

Table 1. Different approximations for $\cos(\pi/16)$.

number	# additions	considered for search
1	0	x
31/32	1	
63/64	1	x
251/256	2	x
2009/2048	3	
4017/4096	3	
8035/8192	4	
16069/16384	3	x

From the data generated to produce Figure 3, we observed that 6 additions are sufficient to multiply by any 20-bit constant. Thus, if $n = 20$, the actual search space is at most of size 7^N (for each constant, between 0 and 6 additions may be chosen, depending on the specific bitwidth). In this paper we consider $n = 16$ for which 5 additions are sufficient; thus, the search space has size of at most 6^N. (In fact, 5 additions are sufficient up to $n = 19$, as has been shown in [6].) Even a small transform, such as $\mathrm{DCT}_8^{(II)}$, requires at least 11 multiplications, which in this case means a search space of size 6^{11}, showing that exhaustive search is impractical even for smaller sized transforms.

Optimization through search. We provide three heuristic search methods that visit only a small subset of the possible bitwidth configurations while trying to find a close-to-optimal solution. As before, n is the maximum bitwidth considered. Evaluation of $\widetilde{T}$ means computing $E(\widetilde{T})$; the threshold that must not be exceeded is $E_{\max}$. A bitwidth configuration for an algorithm means a particular choice of bitwidth for each constant. As explained above, only those

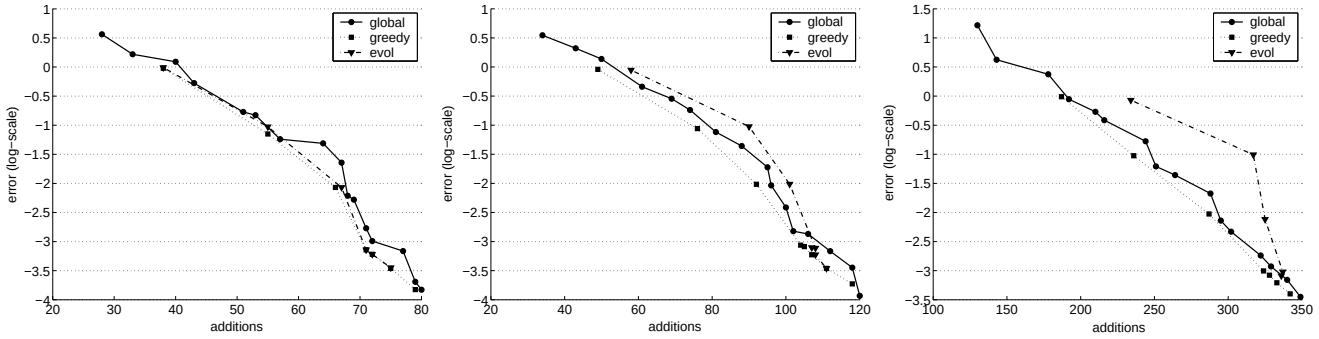

Figure 5. Accuracy cost tradeoffs for $\mathrm{DCT}_8^{(\mathrm{II})}$, $\mathrm{DCT}_8^{(\mathrm{IV})}$, and IMDCT_{18} found using global, greedy, and evolutionary search.

bitwidths that are maximal among all that incur the same cost are considered.

Global search is the simplest method. It assumes that all constants have the same bitwidth $k \in \{0, \ldots, n\}$. Since there are only $n+1$ alternatives, they are each evaluated and the smallest k that still satisfies $E_{\max}$ is the result.

Greedy search (top-down) starts by assigning each constant the maximum bitwidth n. Each constant in turn is now reduced to require one fewer addition and the resulting $\widetilde{T}$ is evaluated. The choice with the least increase in E is chosen. The procedure is repeated until $E_{\max}$ is exceeded.

Evolutionary search optimizes by mimicking the natural process of evolution. In the first step, a random population of algorithm approximations corresponding to constant bitwidth configurations is chosen. Then the population is increased by generating random new individuals and performing *mutations* and *cross-breeding*. Mutation of an algorithm is implemented by changing the bitwidth of one of its constants. Cross-breeding between two approximations is implemented by exchanging the bitwidths of the same constant in each. After expanding the population with mutations and cross-breeding, the fittest individuals (those that require few additions and satisfy the error constraint) are selected to form the next population. The procedure is repeated for a certain number of generations.

EXPERIMENTAL RESULTS

In this section we present experiments that evaluate our approach. All evolutionary searches were conducted using a population size of 75. During each generation, 10 random individuals were inserted, 10 mutations were performed, and 10 pairs were cross-bred. The search terminated after 25 generations. In all global and greedy searches, the constants considered have a maximum bitwidth $n = 16$ (initially, 15 are fractional bits). The application-based error measures for JPEG and MPEG were evaluated by generating code for the reduced precision algorithms with SPIRAL and inserting it into the respective application.

Numerical error measure and search methods. This first set of results characterizes the general relationship between accuracy and cost. The results also assess the relative effectiveness of the different search methods. For this first experiment, we generated custom-optimized implementations for the $\mathrm{DCT}_8^{(\mathrm{II})}$, the $\mathrm{DCT}_8^{(\mathrm{IV})}$, and the IMDCT_{18} with respect to the numerical error measure $E(T) = ||T - \widetilde{T}||_\infty$. For the greedy and evolutionary search, we used multiple error thresholds within the following range: $10^{-4} \leq E_{\max} \leq 1$. The tradeoffs between cost and accuracy are shown in Figure 5. We observe that greedy search generally fared better than global search, which is not surprising since it considers a larger space of possible constant configurations. Evolutionary search finds solutions very close to greedy search for $\mathrm{DCT}_8^{(\mathrm{II})}$, but yields suboptimal results in the other cases. The reason is the difference in the number of multiplications required to compute these transforms—$12, 20$ and 47 for the $\mathrm{DCT}_8^{(\mathrm{II})}$, the $\mathrm{DCT}_8^{(\mathrm{IV})}$, and the IMDCT_{18}, respectively. In the latter two cases, 25 generations are apparently not sufficient for the search to converge. Though some searches perform better than others, all three return reasonable results. The search time in each case was several minutes.

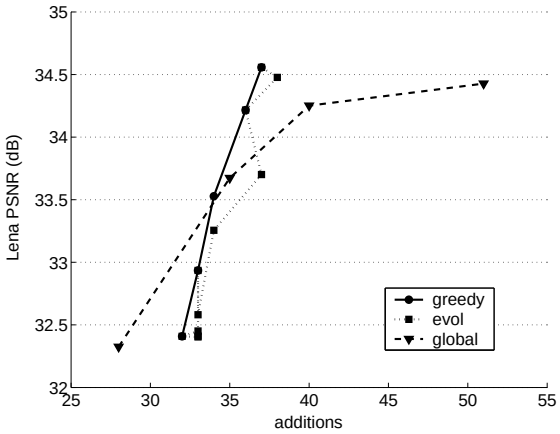

Figure 6. Cost reduction for $\mathrm{DCT}_8^{(\mathrm{II})}$ in JPEG using global, evolutionary, and greedy search.

JPEG. For the second experiment, we custom-optimized the $\mathrm{DCT}_8^{(\mathrm{II})}$ used in JPEG image decoding. As the error measure, we computed the PSNR of the decompressed Lena image (a commonly used DSP reference image), which was com-

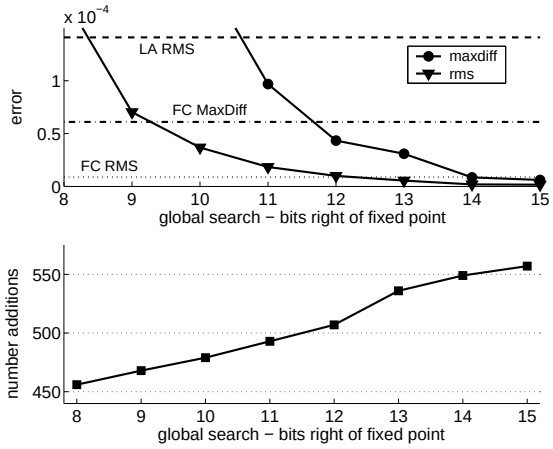

Figure 7. Global search for reducing the $\mathrm{DCT}_{32}^{(\mathrm{II})}$ in MP3 audio decoding with error constraints LA and FC.

pressed using a full-precision $\mathrm{DCT}_8^{(\mathrm{II})}$. For the error threshold, we used different values within the following range: $32 \leq E_{\min} \leq 35$ (recall that higher PSNR is better, hence $E_{\min}$ is used). The results are shown in Figure 6; all three search methods were used. Each search took one hour at most. Greedy and evolutionary search outperform global search for higher PSNR thresholds; global search, in turn, is better for PSNRs less than 33.5 dB. This experiment motivates the use of search methods other than global search, since these other methods are able to tailor the precision of individual constants. We have no explanation for poor performance of greedy and evolutionary search in low PSNR regions.

MP3. For the final experiment, we custom-optimized the $\mathrm{DCT}_{32}^{(\mathrm{II})}$ and IMDCT_{18} in the MP3 audio decoder. We used both the test for full compliance (FC) and the test for limited accuracy (LA) for our error measures (see the explanation in the Accuracy Configuration section regarding LA and FC compliance ratings). We conducted only a global search, which took several minutes. First we custom-optimized only the $\mathrm{DCT}_{32}^{(\mathrm{II})}$, leaving the IMDCT at full precision. Figure 7 shows the result. The RMS (root-mean-square) error drops below the LA threshold for 9-bit precision and above. The FC RMS threshold is satisfied for 13 bits and above; the FC MaxDiff threshold for 12 bits and above. Thus, 13 bits are sufficient for FC (since both FC thresholds are met). A similar experiment for optimizing solely the IMDCT (leaving the DCT at full precision) yields the same bitwidths. Surprisingly, a joint optimization of both transforms (i.e., reducing both concurrently) produces the same bitwidths. The results are summarized in Table 2 together with the costs (BW is the determined global bitwidth/precision; ref BW stands for reference bitwidth). The reference is an implementation with 15 fractional bits per constant. Note that the cost for this reference is also computed using addition chains. CSD would fare considerably worse for this bitwidth as can be seen from Figure 3.

Table 2. Global search for reducing additions in the $\mathrm{DCT}_{32}^{(\mathrm{II})}$ and the IMDCT_{18} in MP3 audio decoding.

	LA		FC		ref BW	
	BW	adds	BW	adds	BW	adds
DCT	9	468	13	536	15	557
IMDCT	9	288	13	329	15	349
DCT + IMDCT	9	756	13	865	15	906

CONCLUSIONS

All successful designs are compromises between the design quality and the design effort. Application specific tuning offers a wealth of optimization opportunities, but at the same time, the tuning effort may be time-consuming and require specialized knowledge not generally available to the designer. This paper considered the problem of minimizing the number of additions, subject to an error threshold, in multiplierless implementations of linear DSP transforms.

The paper presented a fully automatic design generation and optimization framework that addresses this optimization problem at two levels. First, we presented an automatic tool for selecting an algorithm that is robust to low precision fixed point approximations. Next, we presented another automatic tool to tailor the precisions of individual constants to minimize the number of additions while staying below a specified error threshold. In the former tool, high-level domain-aware automation frees the designer from having a detailed knowledge of DSP mathematics; in the latter, efficient feedback-driven exploration replaces the designer in the tedious task of simultaneously optimizing a large number of low-level parameters.

We applied our automatic framework to the optimization of several transforms and multimedia applications. We used our tools to automatically explore a large number of design points and investigated the tradeoff between resource constraints and implementation quality. Overall, our results show that custom design generation and optimization can yield significant cost reduction. The ease of use of this fully automatic framework offers a flexible alternative to static IP design reuse.

ACKNOWLEDGMENTS

The authors wish to acknowledge the support by the National Science Foundation through awards ACR-0234293, SYS-0310941, and ITR/NGS-0325687. Further thanks go to Yevgen Voronenko for helping with the experiments.

REFERENCES

[1] Reinaldo A. Bergamaschi. Behavioral synthesis: An overview. *TR 20944, IBM T.J. Watson Research Center*, Aug 1997.

[2] D.R. Bull and D.H. Horrocks. Primitive operator digital filters. *IEE Proceedings G*, 138(3):401–412, 1991.

[3] Y.-J. Chen, S. Oraintara, T. D. Tran, K. Amaratunga, and T. Q. Nguyen. Multiplierless approximation of

transforms with adder constraint. *IEEE Signal Processing Letters*, 9(11):344–347, 2002.

[4] 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.

[5] F. Franchetti and M Püschel. Short vector code generation for the discrete Fourier transform. In *Proc. IPDPS*, pages 58–67, 2003.

[6] O. Gustafsson, A. G. Dempster, and L. Wanhammar. Extended results for minimum-adder constant integer multipliers. In *Proc. ISCAS*, volume I, pages I–73–I–76, 2002.

[7] N.J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2nd edition, 2002.

[8] Independent JPEG Group. JPEG image compression software. Online. http://www.ijg.org.

[9] ISO/IEC. *Information Technology – Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbits/s – Part 4: Compliance testing*, 1995.

[10] Israel Koren. *Computer Arithmetic Algorithms*. A. K. Peters, 2nd edition, 2001.

[11] R. Leslie. MAD MPEG audio decoder software. Online. http://www.underbit.com/products/mad.

[12] J. Liang and T.D. Tran. Fast multiplierless approximations of the DCT with the lifting scheme. *IEEE Transactions on Signal Processing*, 49(12):3032–3044, 2001.

[13] M. Potkonjak and J. Rabaey. Algorithm selection: A quantitative optimization intensive approach. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 524–32, May 1999.

[14] 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.

[15] S. Roy and P. Banerjee. An algorithm for converting floating-point computations to fixed-point in MATLAB based FPGA design. In *Proc. DAC*, 2004.

[16] C. Shi and Brodersen R. W. Automated fixed-point data-type optimization tool for signal processing and communication systems. In *Proc. DAC*, 2004.

[17] A. C. Zelinski, M. Püschel, S. Misra, and J. C. Hoe. Automatic cost minimization for multiplierless implementations of discrete signal transforms. In *Proc. ICASSP*, 2004.