FACTORED COMPUTING STRUCTURES FOR MULTIPLIERLESS FIR FILTERS USING SYMBOL-SEQUENCE NUMBER SYSTEMS IN LINEAR SPACES

Jeffrey O. Coleman

Naval Research Laboratory
Washington DC
jeffc@alum.mit.edu

Abstract— A general framework is presented here for developing number systems to represent coefficient vectors in an inner-product space. Multiplierless computational structures then follow for computing sums of inner products of data vectors and fixed coefficient vectors. Those structures can often be factored into multiplierless forms having remarkable computational efficiency.

1 INTRODUCTION

An FIR (finite-impulse response) digital filter in direct form can be viewed as a linear combination of some number of scalar data inputs, as an inner product $\langle \mathbf{x}, \mathbf{w} \rangle$ of a data vector $\mathbf{x}$ and some coefficient vector $\mathbf{w}$, or as a sum $\sum_k \langle \mathbf{x}_k, \mathbf{w}_k \rangle$ of inner products of shorter vectors. Filter-and-sum array-pattern synthesis is naturally expressed as such a sum using either a data vector per delay or a data vector per sensor element. More generally, any linear combination can be written as a sum of multidimensional inner products in as many ways as the set of scalar terms required can be partitioned. Choosing such a partition is intimately bound up with choosing a quantization scheme for vector coefficients, with choosing a number system in which to represent the quantized coefficients, and with choosing an efficient computational structure. The topic is too large for one small paper. But let us take the first step here and consider number systems for representing vector coefficients and then computational structures based on those number systems. This amounts to the generalization of the classic multiplierless FIR-filter structure that is based on coefficients expressed in canonical signed-digit (CSD) form. Remarkably, the present development is simplest in a general linear space equipped with an inner product.

2 COEFFICIENT NUMBER SYSTEMS

Creating new terminology freely will streamline discussion: a *representation* in a number system is a finite string of digits representing some coefficient vector, the *value* of the representation, in an inner-product space. A representation has a first digit. Informally,
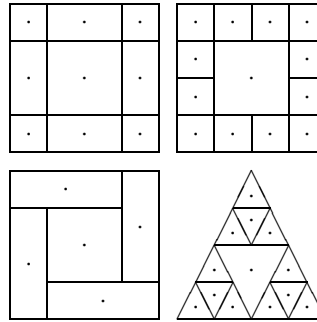
Figure 1: Four example 2D number systems defined by regions $\mathcal{C}$ and their partitions into $\{\mathcal{C}_\alpha\}$ with dots for digit offsets. Center-dot coordinates are zero, others are integers, and radix matrices scale coordinates by simple factors.

the first digit has the same weight for all representations, so the most-significant digit (least-significant digit) is first for ordinary fractions (integers). This order does (does not) follow common custom.

Here $\mathcal{C}$ will denote those vectors in the inner-product space that are the values of representations. The choice of first digit for a representation partitions $\mathcal{C}$ into subsets $\{\mathcal{C}_\alpha : \alpha \in \mathcal{A}\}$, where $\mathcal{A}$ is the digit alphabet. Here digits are not merely symbols but are also functions that embody their meanings.

Definition: *If $\alpha : \mathcal{C} \to \mathcal{C}_\alpha$ is a digit, then $\alpha(\mathbf{r})$ is the value of the representation comprising α followed by the representation of vector $\mathbf{r}$, which is termed a (normalized) remainder.*

Since $\alpha(\mathcal{C}) = \mathcal{C}_\alpha$ by definition of $\mathcal{C}_\alpha$, digit functions are onto.

When each digit function is affine, of form $\alpha(\mathbf{x}) = \mathbf{d} + \mathcal{L}(\mathbf{x})$, a bounded linear function plus an offset, function $\mathcal{L}(\cdot)$ is the digit's *radix* and $\mathbf{d}$ is the digit's *value*. (Otherwise only representations have values, not digits.) If all digits share one radix, it becomes the *number-system radix*. In real N-space $\mathbb{R}^N$ or complex N-space $\mathbb{C}^N$ a linear function is a matrix multiplication $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ and we speak of *radix matrix* $\mathbf{A}$. In scalar spaces this radix becomes scalar also and (in absolute value) is either the same as or the reciprocal of the usual radix, depending on the digit ordering.

Example: *Ordinary terminating nonnegative binary fractions have coefficient values on the real line, a linear space with inner product $\langle a, b \rangle = ab$, and $\mathcal{C}$ comprises rational numbers in interval $[0, 1)$ of form $k/2^n$ with $0 \le k < 2^n$ and $n \in \mathbb{N} = 1, 2, \ldots$ Alphabet*
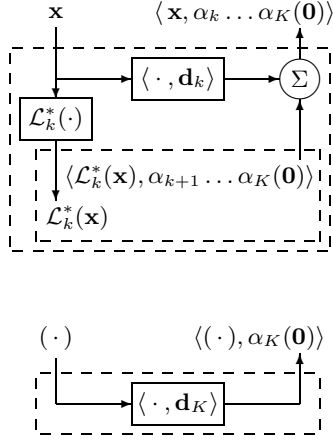
Figure 2: Structural recursion relation (above) and its termination structure (below) for the inner-product of data vector $\mathbf{x}$ with coefficient vector $\alpha_k \ldots \alpha_K(\mathbf{0})$.
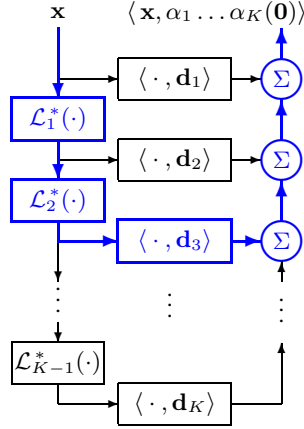
Figure 3: Iterated computational structure implied by the recursion of Fig. 2. The *contribution of digit value* $\mathbf{d}_3$ *is emphasized.*
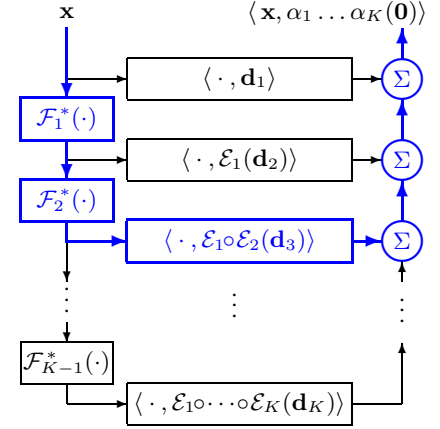
Figure 4: This variant of the Fig. 3 structure sometimes simplifies computation if each $\mathcal{L} = \mathcal{E} \circ \mathcal{F}$ and if each $\mathcal{E}$ operator commutes with all $\mathcal{F}$ operators.

$\mathcal{A}$ contains two digit functions on $\mathcal{C}$:

$$\mathbf{0}(x) = \tfrac{1}{2}\,x \qquad onto\ \mathcal{C_0} = \mathcal{C} \cap \left[0, \tfrac{1}{2}\right)$$
$$\mathbf{1}(x) = \tfrac{1}{2}(x+1)\ \ onto\ \mathcal{C_1} = \mathcal{C} \cap \left[\tfrac{1}{2}, 1\right).$$

Alphabet $\mathcal{A}$ must contain a zero digit with $\mathbf{0}(\mathbf{0}) = \mathbf{0}$.

A composition like $\alpha_1 \circ \alpha_2 \circ \alpha_3 \to \mathcal{C}_{\alpha_1}$, defined by $(\alpha_1 \circ \alpha_2 \circ \alpha_3)(\mathbf{x}) = \alpha_1(\alpha_2(\alpha_3(\mathbf{x})))$ and mapping $\mathcal{C}$ into its subset that has elements with representations beginning with digits $\alpha_1\alpha_2\alpha_3$, is here denoted by $\alpha_1\alpha_2\alpha_3$ for convenience, so $\alpha_1\alpha_2\alpha_3$ represents value $\alpha_1\alpha_2\alpha_3(\mathbf{0})$. Its first digit is α_1.

Example continued: *Numbers conventionally written as finite-length binary strings ".011..." are given by* $\mathbf{0}(\mathbf{1}(\mathbf{1}(\mathcal{C}))) = (\mathbf{0} \circ \mathbf{1} \circ \mathbf{1})(\mathcal{C}) = \mathbf{011}(\mathcal{C})$. *Sequence* $\mathbf{011}$ *represents value* $\mathbf{011}(0) = \tfrac{3}{8}$. *Digit order is conventional, and the radix is* $\tfrac{1}{2}$.

Example: *Those numbers* $\mathcal{C}$ *representable as CSD fractions comprise some dense rational subset of interval* $\left(-\tfrac{2}{3}, \tfrac{2}{3}\right)$. *There are three digit functions:*

$$\mathbf{10}(x) = \tfrac{1}{4}x + \tfrac{1}{2}\ \ onto\ \mathcal{C_{10}} = \mathcal{C} \cap\ \ \left(\tfrac{1}{3}, \tfrac{2}{3}\right)$$
$$\mathbf{0}(x) = \tfrac{1}{2}x \qquad onto\ \mathcal{C_0}\ = \mathcal{C} \cap\ \left(-\tfrac{1}{3}, \tfrac{1}{3}\right)$$
$$\underline{\mathbf{10}}(x) = \tfrac{1}{4}x - \tfrac{1}{2}\ \ onto\ \mathcal{C_{\underline{10}}} = \mathcal{C} \cap \left(-\tfrac{1}{3}, -\tfrac{2}{3}\right)$$

Three-digit sequence $\underline{\mathbf{10}}\ \mathbf{0}\ \mathbf{10}$, *for example, has value*

$$\underline{\mathbf{10}}\ \mathbf{0}\ \mathbf{10}\,(0) = \frac{1}{4}\left(\frac{1}{2}\left(\frac{1}{4} \times 0 + \frac{1}{2}\right)\right) - \frac{1}{2} = -\frac{7}{16}.$$

Considering this as three digits instead of the five simplifies the mathematics considerably. This is a two-radix system.

Example: *For the nonnegative integers, a binary* $\mathcal{A}$ *suffices:*

$$\mathbf{0}(x) = 2x \qquad onto\ the\ even\ nonnegative\ integers$$
$$\mathbf{1}(x) = 2x + 1\ \ onto\ the\ odd\ nonnegative\ integers.$$

Here the set of numbers conventionally written (finitely) in binary as "... 1011" is given by $(\mathbf{1} \circ \mathbf{1} \circ \mathbf{0} \circ \mathbf{1})(\mathcal{C}) = \mathbf{1101}(\mathcal{C})$. *Sequence*

$\mathbf{1101}$ *itself represents value* $\mathbf{1101}(0) = 2(2(2(2 \times 0 + 1)) + 1) + 1 = 8 + 2 + 1$. *Digit order is the reverse of convention, the radix is two, and digit values are zero and one.*

Example: *A dense subset* $\mathcal{C}$ *of interval* $(-16, 16)$ *can be represented with these 13 digit functions (shown in [1]):*

$$\underline{\mathbf{f0}}(x) = x/16 - 15\ onto\ \mathcal{C_{\underline{f0}}} = \mathcal{C} \cap (-14, -16)$$
$$\underline{\mathbf{d0}}(x) = x/16 - 13\ onto\ \mathcal{C_{\underline{d0}}} = \mathcal{C} \cap (-12, -14)$$
$$\underline{\mathbf{b0}}(x) = x/16 - 11\ onto\ \mathcal{C_{\underline{b0}}} = \mathcal{C} \cap (-10, -12)$$
$$\underline{\mathbf{90}}(x) = x/16 - 9\ \ onto\ \mathcal{C_{\underline{90}}} = \mathcal{C} \cap\ \ (-8, -10)$$
$$\underline{\mathbf{70}}(x) = x/16 - 7\ \ onto\ \mathcal{C_{\underline{70}}} = \mathcal{C} \cap\ (-6,\ -8)$$
$$\underline{\mathbf{50}}(x) = x/16 - 5\ \ onto\ \mathcal{C_{\underline{50}}} = \mathcal{C} \cap\ (-4,\ -6)$$
$$\mathbf{0}(x) = x/4 \qquad onto\ \mathcal{C_0}\ = \mathcal{C} \cap\ (-4,\ \ 4)$$
$$\mathbf{50}(x) = x/16 + 5\ \ onto\ \mathcal{C_{50}} = \mathcal{C} \cap\ \ \ (4,\ \ 6)$$
$$\mathbf{70}(x) = x/16 + 7\ \ onto\ \mathcal{C_{70}} = \mathcal{C} \cap\ \ \ (6,\ \ 8)$$
$$\mathbf{90}(x) = x/16 + 9\ \ onto\ \mathcal{C_{90}} = \mathcal{C} \cap\ \ \ (8,\ \ 10)$$
$$\mathbf{b0}(x) = x/16 + 11\ onto\ \mathcal{C_{b0}} = \mathcal{C} \cap\ (10,\ \ 12)$$
$$\mathbf{d0}(x) = x/16 + 13\ onto\ \mathcal{C_{d0}} = \mathcal{C} \cap\ (12,\ \ 14)$$
$$\mathbf{f0}(x) = x/16 + 15\ onto\ \mathcal{C_{f0}} = \mathcal{C} \cap\ (14,\ \ 16).$$

The next two examples show that one-dimensional digit sequences can represent multidimensional values.

Example: *If matrix* $\mathbf{R}$ *on lattice* $\Lambda \subset \mathbb{R}^N$ *yields sublattice* $\mathbf{R}\Lambda \subset \Lambda$, *then family* $[\Lambda/\mathbf{R}\Lambda]$ *of coset representatives contains* $\sigma = \det(\mathbf{R}^T\mathbf{R})^{1/2}$ *elements, call them* $\lambda_1, \ldots, \lambda_\sigma$, *each defining a digit function* ϕ_i *from the lattice to one coset:*

$$\phi_i(\mathbf{x}) = \lambda_i + \mathbf{R}\mathbf{x}.$$

The zero coset $\mathbf{R}\Lambda$ *must be represented by* $\lambda_0 = \mathbf{0}$ *so that* $\phi_0(\mathbf{0}) = \mathbf{R}\mathbf{0}$. *There are many ways to choose the coset representatives* $[\Lambda/\mathbf{R}\Lambda]$, *of course, so this system is not unique.*

When lattice Λ is just the integers and radix matrix $\mathbf{R} = r$ with $[\Lambda/\mathbf{R}\Lambda] = \{0, \ldots, r-1\}$, this lattice system is a digit-order-reversed version of conventional base-r for nonnegative integers.

Actually, if conversion of a negative integer to such a system were attempted using the algorithm discussed below, a digit-reversed r's-complement representation would result, but it would have an infinite number of "sign bits," and so such conversions are not formally permitted. Because this system's $\mathcal{C}$ contains no negatives, we avoid r's-complement.

While a lattice-based system along the same lines can be developed for fractions, a different approach is next considered instead because it offers more flexible number-system design.

Example: *Figure 1 graphically illustrates four separate number systems, each representing some dense subset $\mathcal{C}$ of one of the 2D geometric shapes shown. These shapes are shown partitioned into subsets that when intersected with $\mathcal{C}$ result in the various $\{\mathcal{C}_\alpha\}$. Clockwise beginning with the upper left then, these systems have nine digits, 13 digits, 13 digits, and five digits. We suppose that digit functions are affine with digit values shown by the dots in the figure and with each radix matrix diagonal with the minimal number of negative entries. The center dot is the origin and is in $\mathcal{C}_0$. Each shape can be scaled so that all digit-value dots have integer coordinates.*

The design and performance of number systems of this type are beyond the scope of this paper but follow closely the ideas presented for one-dimensional systems in [1].

2.1 Conversion of Vectors to Representations

We know that representation $\alpha_1 \ldots \alpha_K$ has value $\alpha_1 \ldots \alpha_K(\mathbf{0})$. But how do we go the other way? Given value $\mathbf{x} \in \mathcal{C}$, how do we find representation $\alpha_1 \ldots \alpha_K$ yielding $\alpha_1 \ldots \alpha_K(\mathbf{0}) = \mathbf{x}$?

Conversion is straightforward. The trivial $\mathbf{x} = \mathbf{0}$ has the trivial representation $\mathbf{0}$. Otherwise, nonzero vector $\mathbf{x} \in \mathcal{C}$ has representation $\alpha\beta$ for some digit α and some β that is either a representation or the identity map (null representation), and there is a vector $\mathbf{r} = \beta(\mathbf{0})$ with $\mathbf{x} = \alpha(\mathbf{r})$. The conversion:

Determine α such that $\mathbf{x} \in \mathcal{C}_\alpha$.

Determine $\mathbf{r}$ so that $\alpha(\mathbf{r}) = \mathbf{x}$.

Determine β from remainder $\mathbf{r}$: If $\mathbf{r} = \mathbf{0}$, then β is empty. Otherwise determine β by converting $\mathbf{r}$ recursively.

This tail recursion is straightforwardly made iterative.

The determination of $\mathbf{r}$ in the middle step is unique if digit function α is one-to-one. However, the requirement that the $\{\mathcal{C}_\alpha\}$ be disjoint implies that the first digit of every representation must indeed be unique and hence, by induction, that representations are always unique. So digit functions are necessarily one-to-one and no special precaution is necessary.

Approximation of arbitrary points in the closure of $\mathcal{C}$ is often desired. This simply requires replacing the $\{\mathcal{C}_\alpha\}$ above with their closures and terminating the recursion when approximation is sufficiently close (instead of requiring a zero remainder). This paper will not detail the algorithmic bookkeeping involved, but it is not difficult.

3 COMPUTATIONAL STRUCTURES

Efficient computational structures are derived here first for the inner product of a data vector with a fixed coefficient vector represented with affine digit functions and then for sums of such inner
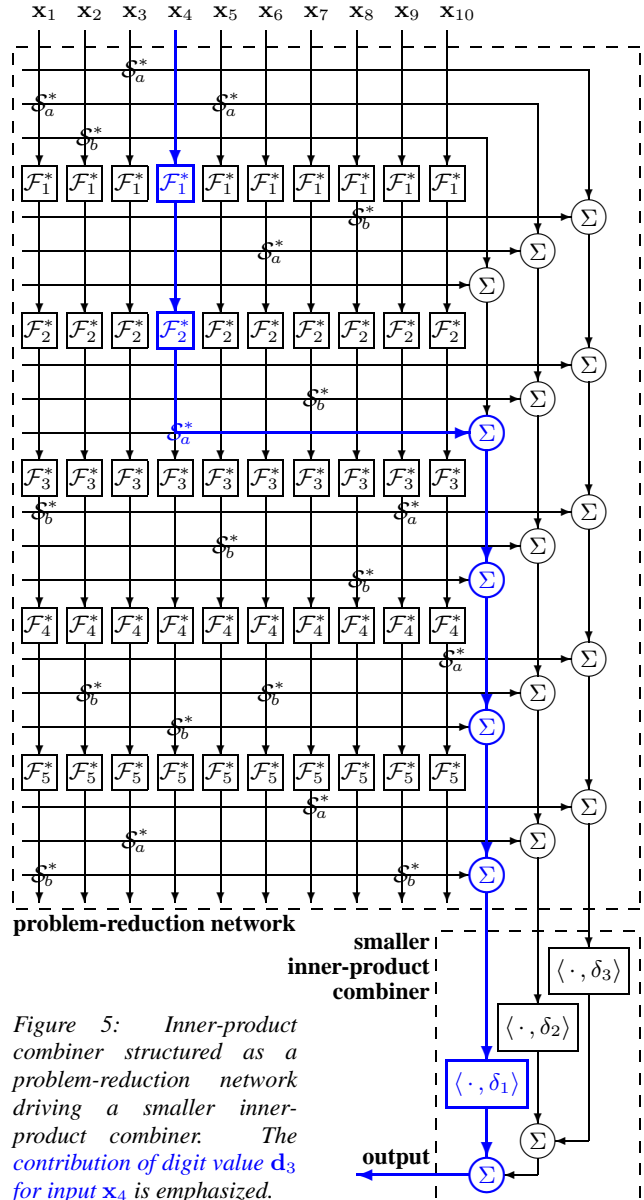

Figure 5: Inner-product combiner structured as a problem-reduction network driving a smaller inner-product combiner. The contribution of digit value $\mathbf{d}_3$ for input $\mathbf{x}_4$ is emphasized.

products. Any format can be used for the data vector and signal paths; two's complement is often appropriate.

Let us begin by expressing the coefficient vector recursively as $\alpha_k \ldots \alpha_K(\mathbf{0}) = \mathbf{d}_k + \mathcal{L}_k(\alpha_{k+1} \ldots \alpha_K(\mathbf{0}))$, terminated by $\alpha_K(\mathbf{0}) = \mathbf{d}_K$, and then taking an inner product with $\mathbf{x}$ on both sides and using the adjoint of the radix operator $\mathcal{L}$ to obtain

$$\langle \mathbf{x}, \alpha_1 \ldots \alpha_K(\mathbf{0}) \rangle = \langle \mathbf{x}, \mathbf{d}_k \rangle + \langle \mathcal{L}_k^*(\mathbf{x}), \alpha_{k+1} \ldots \alpha_K(\mathbf{0}) \rangle.$$

This is realized recursively in Fig. 2 and iteratively in Fig. 3. This is the fundamental structure for computing the inner product of a data vector and a constant coefficient vector.

Recall that for any bounded linear operator $\mathcal{L}(\cdot)$ and its adjoint $\mathcal{L}^*(\cdot)$, inner product $\langle \mathbf{x}, \mathcal{L}(\mathbf{y}) \rangle = \langle \mathcal{L}^*(\mathbf{x}), \mathbf{y} \rangle$ for any $\mathbf{x}, \mathbf{y}$. The adjoint of operator $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ (scaling by a constant matrix) is $\mathbf{x} \mapsto \mathbf{A}^H \mathbf{x}$ with $(\cdot)^H$ denoting conjugate transpose.

330 input variables

problem−reduction network of 166 adders and subtractors

164 variables

problem−reduction
network of
312 adders

64 variables

PR network of
112 adders
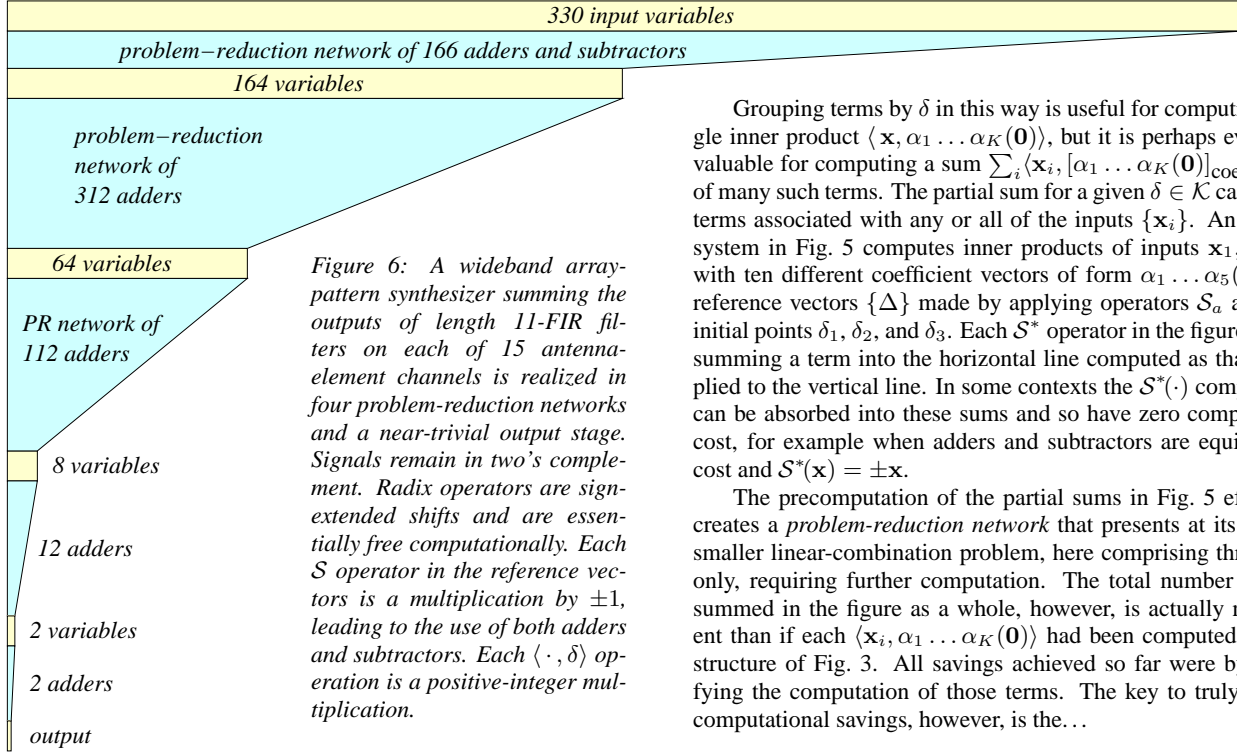
8 variables

12 adders

2 variables

2 adders

output

Figure 6: A wideband array-pattern synthesizer summing the outputs of length 11-FIR filters on each of 15 antenna-element channels is realized in four problem-reduction networks and a near-trivial output stage. Signals remain in two's complement. Radix operators are sign-extended shifts and are essentially free computationally. Each $\mathcal{S}$ operator in the reference vectors is a multiplication by ± 1, leading to the use of both adders and subtractors. Each $\langle \cdot , \delta \rangle$ operation is a positive-integer multiplication.

Some radices $\mathcal{L}_k^*(\cdot)$ can be usefully factored. In the triangular system in Fig. 1, for example, only three radix matrices

$$\frac{1}{2}\left(\begin{array}{cc} 1 & 0 \\ 0 & -1 \end{array}\right) \qquad \frac{1}{4}\left(\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array}\right) \qquad \frac{1}{4}\left(\begin{array}{cc} 1 & 0 \\ 0 & -1 \end{array}\right)$$

are used. In some settings the scalar factors represent simple sign-extended right shifts that are free of computational cost while negations of two's complement data are more expensive.

Generalizing, suppose each radix takes form $\mathcal{L} = \mathcal{E} \circ \mathcal{F}$, where linear operators $\mathcal{E}$ and $\mathcal{F}$ are computationally more $\mathcal{E}$xpensive and less expensive ($\mathcal{F}$ree) respectively, and suppose each operator $\mathcal{E}$ commutes with all $\mathcal{F}$ operators. The $\mathcal{E}$ operators can then be moved out of the real-time computation path and into the inner products where they can be precomputed. The structure in Fig. 3 becomes the variant structure in Fig. 4.

The radix factoring just discussed transformed one system with many inner products of form $\langle \mathbf{y}, \Delta \rangle$ into another, sometimes enlarging the number of distinct reference vectors Δ in the process. Still, any particular Δ will generally appear in many terms, and the partial sum of those terms can be computed efficiently using the linearity of the inner product: $\sum_k \langle \mathbf{y}_k, \Delta \rangle = \langle \sum_k \mathbf{y}_k, \Delta \rangle$. So actual inner products need be computed just once for each distinct reference vector Δ used.

Often it further happens that there is useful structure to the family of reference vectors. For example, for each Δ used, $-\Delta$ may well be used also. Such relationships offer further savings. Generalize by supposing that each reference vector Δ is of form $\mathcal{S}(\delta)$, where bounded linear operator $\mathcal{S}$ has an adjoint $\mathcal{S}^*$ that is computationally simpler (like negation) than the original $\langle \cdot , \Delta \rangle$ operator and with each vector δ from some small set $\mathcal{K}$ of initial points. Then the partial sum of those terms that share a common δ is $\sum_k \langle \mathbf{y}_k, \mathcal{S}_k(\delta) \rangle = \sum_k \langle \mathcal{S}_k^*(\mathbf{y}_k), \delta \rangle = \langle \sum_k \mathcal{S}_k^*(\mathbf{y}_k), \delta \rangle$.

Grouping terms by δ in this way is useful for computing a single inner product $\langle \mathbf{x}, \alpha_1 \ldots \alpha_K(\mathbf{0}) \rangle$, but it is perhaps even more valuable for computing a sum $\sum_i \langle \mathbf{x}_i, [\alpha_1 \ldots \alpha_K(\mathbf{0})]_{\text{coefficient } i} \rangle$ of many such terms. The partial sum for a given $\delta \in \mathcal{K}$ can include terms associated with any or all of the inputs $\{\mathbf{x}_i\}$. An example system in Fig. 5 computes inner products of inputs $\mathbf{x}_1, \ldots, \mathbf{x}_{10}$ with ten different coefficient vectors of form $\alpha_1 \ldots \alpha_5(\mathbf{0})$ using reference vectors $\{\Delta\}$ made by applying operators $\mathcal{S}_a$ and $\mathcal{S}_b$ to initial points $\delta_1, \delta_2,$ and δ_3. Each $\mathcal{S}^*$ operator in the figure denotes summing a term into the horizontal line computed as that $\mathcal{S}^*$ applied to the vertical line. In some contexts the $\mathcal{S}^*(\cdot)$ computations can be absorbed into these sums and so have zero computational cost, for example when adders and subtractors are equivalent in cost and $\mathcal{S}^*(\mathbf{x}) = \pm \mathbf{x}$.

The precomputation of the partial sums in Fig. 5 effectively creates a *problem-reduction network* that presents at its output a smaller linear-combination problem, here comprising three terms only, requiring further computation. The total number of terms summed in the figure as a whole, however, is actually no different than if each $\langle \mathbf{x}_i, \alpha_1 \ldots \alpha_K(\mathbf{0}) \rangle$ had been computed with the structure of Fig. 3. All savings achieved so far were by simplifying the computation of those terms. The key to truly massive computational savings, however, is the...

4 CENTRAL POINT OF THIS PAPER

...that an FIR filter or array-pattern synthesizer or other required sum of inner products in an arbitrary inner-product space can be computed with a cascade of problem-reduction stages, each based on a number system chosen for that stage in a tradeoff between degree of reduction and computational cost. One-dimensional experiments using simple number systems easily realized startling overall computational savings [2]. The structure in Fig. 6, for example, requires 62% fewer adders and subtractors than the classic CSD-based equivalent. In another example, an FIR filter with a real chirp impulse response 3000 samples in length (a matched filter for a hypothetical pulse-compression radar) was easily realized with -95 dB rms approximation error using less than two add or subtract operations per coefficient, a reduction of approximately 60% relative to the usual CSD method.

The hope and expectation is that multidimensional formulations developed on the more-general theory base presented here will yield significant further savings, as will optimization within each problem-reduction network to eliminate common subexpressions. The work continues.

REFERENCES

[1] J. O. Coleman, "Express Coefficients in 13-ary, Radix-4 CSD to Create Computationally Efficient Multiplierless FIR Filters," in *Proc. European Conf. on Circuit Theory and Design* (Espoo, Finland), Aug. 2001.

[2] J. O. Coleman, "Cascaded Coefficient Number Systems Lead to FIR Filters of Striking Computational Efficiency," in *Proc. 2001 Int'l IEEE Conf. on Electronics, Circuits, and Systems* (Malta), Sept. 2001.