AN EFFICIENT ALGORITHM FOR THE MULTIPLIERLESS REALIZATION OF 2-D LINEAR TRANSFORMS

Arda Yurdakul

Boğaziçi University
Computer Engineering Department,
Bebek 80815, ISTANBUL, TURKEY
yurdakul@boun.edu.tr

ABSTRACT

In this work, an algorithm is developed for the automatic generation of multiplierless architectures for two dimensional linear transforms. Though there are a number of algorithms developed for this purpose, the algorithm presented in this paper outperforms the previous ones: When regular CSD (canonic-signed-digit) representation is used for the quantization of scalars in the two dimensional coefficient matrix, architectures employing 20%-60% fewer adders than the origianl architectures. This is improved by an additional 15-25% if the recently-introduced CSD-4 representation is used for the quantization of scalars.

1. INTRODUCTION

Hardware realization of DSP systems is quite common in today's technology. Development of design automation tools to realize this kind of hardware from the system specification is an active research area.

Most DSP algorithms are based on linear transforms. A linear transform can be viewed as the multiplication of a set of inputs with a set of coefficients to have a set of outputs. The coefficients can be real and/or imaginary, depending on the properties of the transform.

Constant coefficient multiplications have been realized using solely a bank of adders (and/or subtracters) with hardwired-shifting operations for the last decade. The basic reason for introducing this solution in place of multipliers was the relatively immense area covered by the multipliers at those times. Besides, multipliers are always known to be slow devices and adder-bank solutions have been much faster than the ones with multipliers. However, improvements in VLSI technology have provided the hardware designers an almost infinite area on the chip. Today it is not surprising to have 100,000 multipliers operating concurrently on a single chip [1]. It should be noted that concurrency improves the speed performance of slow multipliers. Besides, coefficients can be easily updated when multipliers are used in a system. So, the benefits of multiplierless realizations can easily be questioned. However, adder-bank solutions are still very power-efficient. Low power is an essential issue in battery-powered devices. In addition, in near future, it will be quite common to program the hardwired connections of the adder-banks thanks to the development of reconfigurable systems that is expected to be another dimension in computing [2]. This will give the designer a flexibility to update the coefficients whenever it is necessary.

Multiplierless realization of some transforms has already been automated. These tools are basically developed for one- and two-dimensional FIR transforms Some of these tools basically start from the system specifications and generate SOPOT coefficients of the whole two dimensional system [3]-[4]. In the remaining tools, the coefficients that are already generated by some other systems are taken and the multiplierless realization of the system for the user-defined coefficient precision is produced [5], [6]. The algorithm presented in this paper is in this second category. This algorithm produces smaller architectures in CSD and CSD-4 [7] in very short run-times: When regular CSD (canonic-signed-digit) representation is used for the quantization of scalars in the two dimensional coefficient matrix, the architectures produced by the new algorithm presented here employs 20%-60% fewer adders than the original adder-based implementations.Quantizing coefficients using CSD-4 base improves the results up to 70%.

The following section sets the theoretical background of the developed algorithm by pointing out that the problem of realizing a 2-D linear transform with minimum number of adders is an NP-complete problem. In Section 3, the algorithm is explained. The final section presents the experimental results and concludes the work.

This work was supported throughout the Scientific Research Projects program of Boğaziçi University

0-7803-8116-5/02/$17.00 ©2002 IEEE.

2. THEORETICAL BASE OF THE ALGORITHM

A two-dimensional linear transform can be written as

$$\mathbf{y} = \mathbf{K}_{M\times N}\mathbf{x} \tag{1}$$

where $\mathbf{K}$ is the coefficient matrix, $\mathbf{x}$ is the input vector, and $\mathbf{y}$ is the output vector. Coefficent matrix can be written as the combination of P linearly independent sub-matrices, i.e.

$$\mathbf{K} = \sum_{p}^{P} \alpha_p \mathbf{K}_p \tag{2}$$

where α_p is the scaling constant for the sub-matrix $\mathbf{K_p}$. Each sub-matrix can be viewed as a system of row vectors,

$$\mathbf{K}_p = \begin{bmatrix} \mathbf{k}_{p_1}^T \\ \mathbf{k}_{p_2}^T \\ \vdots \\ \mathbf{k}_{p_M}^T \end{bmatrix} = \sum_{m}^{M} 1_m \mathbf{k}_{p_m}^T \tag{3}$$

Here, $\mathbf{k}_{p_m}$ is the m'th row of $\mathbf{K}_p$ and 1_m is the column vector whose m'th entry is 1 and the remaining entries are zero. Using the rows of all sub-matrices, a set of linearly independent row vectors, $\mathbf{k}_r^T$, can be formed to span the space of row vectors. Assume that the cardinality of this set is $R \leq P.M$. Then $\mathbf{k}_{p_m}^T$ is given by

$$\mathbf{k}_{p_m}^T = \sum_{r}^{R} \beta_{r,p_m} \mathbf{k}_r^T \tag{4}$$

The scaling coefficient β_{r,p_m} is nonzero if and only if $\mathbf{k}_r^T$ must be present in the formation of $\mathbf{k}_{p_m}$.

Using equations 3 and 2, the coefficient matrix can be rewritten as

$$\mathbf{K} = \sum_{m}^{M} 1_m \underbrace{\sum_{p}^{P} \alpha_p \mathbf{k}_{p_m}^T}_{\mathbf{k}_m^T} \tag{5}$$

Each row of the coefficient matrix, $\mathbf{k}_m^T$, can also be written using the combination of S linearly independent row vectors, $\mathbf{k}_s^T$.

$$\mathbf{k}_m^T = \sum_{s}^{S} \delta_{sm} \mathbf{k}_s^T \tag{6}$$

Here, δ_{sm} is a binary variable which is 1 if and only if $\mathbf{k}_s^T$ must be present for the final implementation of the m'th row of the coefficient matrix $\mathbf{K}$. Hence, combining Eq. 5 with the equations 4 and 6, the most general form for computing $\mathbf{K}$ can be obtained:

$$\mathbf{K} = \sum_{m}^{M} 1_m \sum_{r}^{R} \sum_{s}^{S} \delta_{sm} \left(\sum_{s}^{S} \alpha_s \beta_{r,s_m} \mathbf{k}_r^T \right) \tag{7}$$

In the above equation, the variables are δ_{sm}, α_s, β_{r,s_m} and $\mathbf{k}_r^T$. Therefore, minimum number of adders to realize $\mathbf{K}$ will be determined by the appropriate choice of these variables. Eq. 7 also hints that the "minimum number of adders for 2-D linear transforms" problem for can be modelled using a nonlinear mixed integer programming model. The selection of linearly independent $\mathbf{k}_r^T$ vectors also shows that the problem is NP-complete (the proof is detailed and omitted in this paper). In the following section, heuristics are proposed to solve this problem is in a short time for near-optimality.

3. THE ALGORITHM

The adder minimization algorithm mainly consists of the following steps:

- Coefficient quantization
- Matrix decomposition (i.e., implementation of Eq. 2)
- Basis vector extraction (i.e., extract $\mathbf{k}_r^T$)
- Adder minimization
- Rowwise minimization (i.e. implementation of Eq. 6)

Below, there will be brief explanations for each step:

1. Coefficient quantization: The matrix entries quantized with a given representation type for a given wordlength.

2. Matrix Decomposition: The matrix decomposition is based on extracting common patterns in the quantized coefficients. To achieve this aim, an algorithm that realizes multiplierless 1-D systems efficiently in a very short runtime [8] is used in this study, because it is assumed that any algorithm that minimizes the number of adders used in the system should maximize the common terms between different coefficients. The 1-D algorithm of [8] produces a tree of minimum adders. Obviously, the tree has at most $M.N$ leaves because there are at most $M.N$ different and quantized coefficients in $\mathbf{K}$ (Eq. 1). All paths from a leaf to the root will form a sub-tree. Assume that Υ is the forest of such sub-trees for all leaves. The intersection of sub-trees in a sub-forest of Υ is the common pattern between the related leaves of the original tree. As the aim is the minimization of the number of adders in the whole system, then the subforest satisfying the following two conditions simultaneously must be selected.

- The cardinality of the selected sub-forest $\Upsilon^* \subseteq \Upsilon$ is maximum.
- The number of adders on t^*, the intersection of the sub-trees, in Υ^* is maximum.

Each leaf of t^* is the α scalar of Eq. 2. Assume that t^* has $A \leq P$ leaves. Then Eq. 2 can be written as

$$\mathbf{K} = \sum_{a}^{A} \alpha_a \mathbf{K}_a + \mathbf{K}' \tag{8}$$

Here, zero entries of $\mathbf{K}_a$ stand for the coefficients whose sub-trees are not in Υ^*. The nonzero entries of $\mathbf{K}_a$ stand for the shifting and/or negation operations. $\mathbf{K}'$ is the residual matrix which consists of all entries that are not in t^*. The residual matrix is regarded as the actual coefficient matrix $\mathbf{K}$ and the above process is repeated until t^* is made up of solely the root.

The aim of this iterative process is to realize Eq. 2. Yet, since the sub-matrices must be linearly independent, it is expected that $\mathbf{K}_p$ must be unique. Besides, since the objective is minimizing the number of adders, then α_p should also be unique. The iterative algorithm cannot always satisfy these two conditions. Hence, if there are more than one α_p (or $\mathbf{K}_p$) with the same value, then the related $\mathbf{K}_p$'s (or α_p's) must be added up and the iterative process must be repeated until uniqueness of $\mathbf{K}_p$ and α_p is satisfied. At the end of this process, there are $P-1$ different α_p terms which are different than 1, and one α_p term which is one. The entries of $\mathbf{K}_p$ terms are only shifting and/or negation operations.

3. Basis vector extraction: Iteratively unique rows are extracted from all sub-matrices, i.e. $\mathbf{K}_p$'s. By the term "unique" it is meant that the rows that can be realized by merely scaling a row that is already a member of the basis vector set are ignored. It is obvious that the basis obtained in this manner is not always optimum because there might be common terms between rows and these common terms can be shared to reduce total number of adders. Hence, a slightly modified version of 1-D algorithm of [8] can be used. Note that each $\mathbf{K}_p$ is not necessarily a zero-one matrix, none of the previously designed 1-D algorithms except [8] can be used to solve this problem, because in [8], symbols are used in place of actual digit values which are not necessarily one or zero. In [8], the symbol is the "two-term" which has exactly two nonzero digits separated by a number of zero digits. A two-term is can be regarded as a mask sweeping over digits of a number to find matching patterns. The two-terms with highest number of matching patterns is selected. The patterns represented by the selected two-term is replaced with a symbol and the process continues until a single symbol is found to represent the number. In the modified version of [8] used in basis vector extraction, "two-term" is modified as a mask sweeping over the rows of matrices to find the "matching pattern"s. If a two-term in a row is the shifted and/or negated version of another two-term in another row, then these two two-terms are regarded as matching. Note that columns of matching patterns must be the identical. The two-term with highest number of matching patterns appears as a new column introduced to all rows. Note that the signal on the newly introduced column is equivalent to the addition of two inputs (i.e. columns) that are used to form the selected two term. The entries of this new column is nonzero wherever a matching pattern appears. The patterns represented by the selected two terms are simply erased. The process continues until no two-term with more than one matching pattern appears. The set of all symbols and the remaining rows with unmaching patterns constitute the set of R basis vectors, $\mathbf{k}_r^\tau$. The entries of newly added columns are the scalars β_{r,p_m}. Hence, Eq. 4 is realized. Note that each basis vector $\mathbf{k}_r^\tau$ is realized with an adder bank. The number of adders per $\mathbf{k}_r^\tau$ is "symbol depth", d_r, which is a term defined as the number of symbols used to realize another symbol.

4. Adder minimization: Adder minimization is done over $\alpha_p.\beta_{r,p_m}$ by applying a 1-D adder minimization algorithm such as [8] in two different directions independently, to obtain $(\alpha_p.\beta_{r,p_m}\mathbf{k}_r^\tau)\,\mathbf{x}$ where $\mathbf{x}$ is the input vector that is enriched with newly added symbols :

- Horizontal direction: Each row $\mathbf{k}_r^\tau$ can be realized by a bank of adders. The inputs to the bank are the nonzero column entries for each row. The output of the final adder of the bank, $\mathbf{k}_r^\tau\mathbf{x}$, is regarded as an input to the set of coefficients $\alpha_p.\beta_{r,p_m}$. After applying the 1-D adder minimization algorithm, each leaf of the generated adder tree will be the scaled version of the $\mathbf{k}_r^\tau$ by $\alpha_p.\beta_{r,p_m}$.

- Vertical direction: For each column, the $\alpha_p.\beta_{r,p_m}$ scalars are picked and grouped. The column is regarded as an input to the picked scalars and 1-D adder minimization algorithm is applied. Each leaf is the scaled version of the input signal on that column. Leaves of the trees of all columns are used to realize each row, $\mathbf{k}_r^\tau$.

The number of adders obtained from two directions need not be identical. Besides, depending on the initial coefficients in $\mathbf{K}$, one of the two directions might produce better results. The direction with fewer adders is selected as the final implementation style.

5. Rowwise minimization: To realize Eq. 1, all scaled $\mathbf{K}_p\mathbf{x}$'s must be added up. Note that scaled version of each row of the sub-matrices has already been realized in previous step. While adding up the rows, some common terms might be shared between different rows of the final $\mathbf{K}$. At this stage, for each scaled input $(\alpha_p.\beta_{r,p_m}\mathbf{k}_r^\tau)\,\mathbf{x}$ for each row in each $\mathbf{K}_p$ is represented by a unique symbol.1-D algorithm of [8] is applied to realize the rowwise addition with a minimum adder tree. Note that this procedure is identical to realize Eq. 6.

After these five steps is completed, the Eq. 1 is realized with minimum number of adders.

Table 1. Experimental Results: Number of Adders

Exp	*Org*	*CSD* W	*CSD-4* $\lceil (W+1)/2 \rceil$	*CSD-4* $\lceil (W+3)/2 \rceil$
DCT_8	200	86	43	52
DCT_12	264	110	64	98
DCT_16	344	154	122	118
DCT_24	536	211	148	148
pp_8	94	55	12	45
pp_12	98	81	49	80
pp_16	151	112	88	102
pp_24	196	164	136	144

Table 2. Experimental Results: Percent Gain

Exp	*CSD* W	*CSD-4* $\lceil (W+1)/2 \rceil$	*CSD-4* $\lceil (W+3)/2 \rceil$
DCT_8	57	79	74
DCT_12	58	76	63
DCT_16	55	65	66
DCT_24	61	72	72
pp_8	41	87	52
pp_12	17	50	18
pp_16	26	42	32
pp_24	16	31	27

4. EXPERIMENTAL RESULTS AND CONCLUSION

The algorithm is run on some examples in the literature. These results are tabulated in Tables 1 and 2. The first table gives the actual number of adders after the algorithm is executed while the second table is used to demonstrate percent gain over the original adder count. As examples, discrete cosine transform (DCT) and an eight-branch polyphase filter presented (pp) in [9] are run for four different wordlengths, i.e 8, 12, 16, and 24 bits. However, quantization in CSD-4 is done in base-4, unlike the ordinary CSD. Besides, the author of [7] states that the CSD-4 dynamic range is nearly 4:6 dB plus 12 dB per CSD-4 digit kept as against nearly 6 dB plus 6 dB per CSD digit kept for ordinary CSD. Therefore, to make a meaningful comparison between a system implemented in ordinary CSD and CSD-4, it seems that half of the wordlengths used in CSD should be used while implementing a system in CSD-4. Yet, experimental results have shown that the system quality of ordinary CSD in a CSD-4 implementation can be retained if and only if one uses at least one more digit than the half of the bits CSD-4 for coefficient quantization. Superior results are obtained if two more digits than the half of the bits CSD-4 are used. Therefore different wordlengths for different representation schemes are shown in the tables as W for ordinary CSD wordlength, $\lceil (W+1)/2 \rceil$ for the implementation with CSD-4 with quatization noise closer to ordinary CSD implementation, and $\lceil (W+3)/2 \rceil$ for the implementation with CSD-4 with quantization noise superior than ordinary CSD implementation.

As it can be observed from both tables, the new algorithm improves the results in ordinary CSD implementations. The improvement is much more notable in CSD-4 implementations. The above explanation implies that CSD-4 adders will be much smaller than the ordinary CSD adders because smaller wordlengths in CSD-4 are sufficient to obtain a similar or superior performance than ordinary CSD. In addition, fewer number of adders are used in CSD-4. Therefore actual gain in area in CSD-4 implementation will be much more than the ordinary CSD implementation.

5. REFERENCES

[1] R. Brodersen, "Keynote: Why we need a custom chip- in- a- day design methodology," http://www.mseconference.org/confplan.html.

[2] P. Schaumont, I. Verbauwhede, K. Keutzer and M. Sarrafzadeh, "A quick safari through the reconfiguration jungle," http://www.dac.com/39th/talkindex.html.

[3] S. C. Pei and S. B. Jaw, Efficient design of 2-D multiplierless FIR filters by transformation, IEEE T-CAS, vol. 34, pp. 436-438, April 1987.

[4] W. S. Lu, "Design of 2-D filters with power-of-two coefficients: A semidefinite programming approach," *ISCAS'01*, pp. (II-549)-(II-552), Australia, May 2001.

[5] M. Potkonjak, M. B. Srivastava, and A. P. Chandrakasan, "Multiple constant multiplications: Efficient and versatile framework and algorithms for exploring common subexpression elimination," *IEEE T-CAD.*, vol. 15, pp. 151-165, Feb. 1996.

[6] H. T. Nguyen and A. Chatterjee, "Number-splitting with shift-and-add decomposition for power and hardware optimizations in linear DSP synthesis", *IEEE T-VLSI*, pp 419-423, Aug. 2000.

[7] J. O. Coleman, "Express coefficients in 13-ary, radix-4 to create computationally efficient multiplierless FIR filters", *Proc. of ECCTD'01*, Finland, Aug. 2001.

[8] A. Yurdakul ve G. Dündar, "A fast and efficient algorithm for the multiplierless realization of linear DSP transforms," *IEE Proc.-Circuits, Devices and Systems*, accepted.

[9] J. O. Coleman, J. J. Alter, and D. P. Scholnik, "FPGA architecture for Gigahertz-sampling wide-band IF-to-baseband conversion," in *Proc. Int'l Conf. on Signal Processing Applications and Tech-nology (ICSPAT 2000)*, Dallas TX, Oct. 2000.