PARALLEL IMPLEMENTATION OF MULTIFILTERS

Niranjan Damera-Venkata and Brian L. Evans

Embedded Signal Processing Laboratory
Dept. of Electrical and Computer Engineering
The University of Texas at Austin, Austin, TX 78712-1084 USA
{damera-v,bevans}@ece.utexas.edu

ABSTRACT

A multifilter is a filter with matrix-valued coefficients, and is used in the processing of vector-valued signals, e.g. color images. Convolution becomes a vector sum of matrix-vector multiplication. In this paper, we efficiently implement a multifilter as a parallel combination of *scalar* filters. Each scalar filter works on one component of the input vector signal, which increases processing speed by the dimension of the vector-valued signal. This means that by using N processors, the throughput is increased by a factor of N while the total memory usage remains unchanged. We also present a frequency-domain analysis of the filtering.

1. INTRODUCTION

Vector-valued signals such as color images often have significant correlation among their components. Applying scalar filters to each component ignores the correlation. Multifilters [1] are filters with matrix-valued coefficients that can take into account the intercomponent correlation of the vector signal being processed. Multiwavelet filter banks [2, 3, 4] are the extension of conventional scalar filter banks to the case of vector-valued signals. These filter banks employ multifilters as the analysis and synthesis filters. Apart from signal compression [5], multifilters have been used in digital image halftoning [6] to improve visual quality over conventional scalar-valued filters.

Multifilters have high implementation complexity even on programmable digital signal processors (DSPs), which are optimized for matrix-vector multiplications due to their single-cycle multiply-accumulate instructions [7]. In [4], the authors derive efficient polyphase implementations of multifiltering followed by downsampling. The filters in the polyphase implementation are themselves multifilters.

This research was supported by a US NSF CAREER Award under grant MIP-9702707.

In this paper, we express the operation of a multifilter (not followed by a downsampling operation) in terms of a conventional scalar filters operating on the the individual components of the vector-valued signal. Section 2 defines the multifilter in the time domain. Sections 3 analyzes the direct implementation of one-dimensional multifilters on parallel programmable DSPs. Section 4 derives an efficient implementation of a one-dimensional multifilter in terms of scalar filters. This efficient implementation is well suited for programmable DSPs as well as VLIW DSPs and general-purpose processors with Single Instruction Multiple Data (SIMD) extensions. Section 5 compares the two parallel implementations. Section 6 extends the more efficient parallel implementation to multiple dimensions. Section 7 concludes the paper.

2. NOTATION

In this paper, $\vec{\mathbf{x}}(\cdot)$ represents the input N-vector valued signal to be filtered. $\hat{\vec{\mathbf{X}}}(\cdot)$ represents the z-transform of the vector-valued signal sequence

$$\hat{\vec{\mathbf{X}}}(z) = \sum_m \vec{\mathbf{x}}(m) z^{-m} \tag{1}$$

The multifilter (which is assumed to be causal) will be denoted by the $N \times N$ matrix valued sequence $\tilde{\mathbf{H}}(\cdot)$. $\hat{\tilde{\mathbf{H}}}(\cdot)$ represents the z-transform of the matrix-valued multifilter sequence defined by

$$\hat{\tilde{\mathbf{H}}}(z) = \sum_m \tilde{\mathbf{H}}(m) z^{-m} \tag{2}$$

A multifilter with M $N \times N$ matrix coefficients can be expressed compactly as the $N \times NM$ matrix

$$\tilde{\mathbf{\Gamma}} = \left[\tilde{\mathbf{H}}(0) \mid \tilde{\mathbf{H}}(1) \mid \ldots \mid \tilde{\mathbf{H}}(M-1)\right] \tag{3}$$

where $\tilde{\mathbf{H}}(0) \cdots \tilde{\mathbf{H}}(M-1)$ are the coefficients. The filtering operation of a multifilter is defined by matrix-vector

convolution is given by

$$\vec{\mathbf{y}}(m) = \sum_{k=0}^{M-1} \tilde{\mathbf{H}}(k)\vec{\mathbf{x}}(m-k) \tag{4}$$

where $\vec{\mathbf{y}}(\cdot)$ represents the N-vector valued output sequence. In the z domain the matrix-vector convolution becomes a linear transformation by an $N \times N$ transformation matrix given by

$$\hat{\vec{\mathbf{Y}}}(z) = \hat{\tilde{\mathbf{H}}}(z)\hat{\vec{\mathbf{x}}}(z) \tag{5}$$

For a scalar signal $x(m)$, we will denote the z-transform by $X(z)$ as usual.

3. PARALLEL PROGRAMMABLE DSPS

A direct implementation of (4) requires N separate multiplication-accumulations to compute each component of a single matrix-vector multiplication. The N rows of the matrix multiply the same N vector in a matrix-vector multiplication. By parallelizing each row of computation, the data in the vector would have to be shared among the processors, and the resulting individual vector dot products would have to be synchronized. The input data is typically put into N circular buffers [7] of size M each to take advantage of the hardware support for circular addressing on the DSP. The filtering would involve sharing of the circular buffers among the parallel processors. This means that synchronization among the processors must be performed within the loops, which will waste precious clock cycles. Therefore, the naive (direct) implementation of (4) requires MN^2 multiplication-accumulations performed sequentially, and does not make efficient use of multiple DSPs even if they are available.

4. PARALLEL BLOCK FILTERING

In the multifilter operation in (4), the output vector at each vector sample location m is computed by taking N linear combinations of *all* of the signal vector components in the set

$$\mathcal{S}_m = \{x_i(m-k), i = 0 \cdots N, k = 0 \cdots M\} \tag{6}$$

The overall filtering may be expressed in terms of the matrix $\tilde{\mathbf{\Gamma}}$ as

$$\vec{\mathbf{y}}(m) = \tilde{\mathbf{\Gamma}}\vec{\mathbf{x}}'(m) \tag{7}$$

where

$$\vec{\mathbf{x}}'(m) = [x_0(m), x_1(m) \cdots x_N(m-M+1)]^T \tag{8}$$

We can reorder the vector-valued components of $\vec{\mathbf{x}}'(m)$ in (7) to create a scalar-valued sequence $x''(\cdot)$ by

$$\begin{aligned} x''(0) &= x_0(0) \\ x''(1) &= x_1(0) \\ &\vdots \\ x''(r) &= x_{r \bmod N}(\lfloor \tfrac{r}{N} \rfloor) \end{aligned}$$

In many situations, the vector-valued signals are produced in "blocks" of an input scalar-valued signal via the process described above. The samples of the nth component of the output signal vector sequence $\vec{\mathbf{y}}(\cdot)$ may be recovered exactly by filtering the scalar sequence $x''(\cdot)$ with the filter $h^{(n)}(\cdot)$, with the additional constraint that only every Nth output be retained. The values of $h^{(n)}(\cdot)$ are formed by reversing the nth row of the matrix $\tilde{\mathbf{\Gamma}}$, These operations may are represented by

$$y_n(m) = \sum_{k=0}^{MN-1} h^{(n)}(k)x''(Nm+N-1-k) \tag{9}$$

In the frequency domain, this operation may be expressed as

$$Y_n(z) = (\downarrow N)\left(H^{(n)}(z)X''(z)z^{(N-1)}\right) \tag{10}$$

Filtering followed by subsampling may be implemented efficiently using the polyphase decomposition [8, 1]. If we consider each of the N components of the input signal vector, then the type-II polyphase decomposition of the filter is

$$H^{(n)}(z) = \sum_{i=0}^{N-1} H_i^{(n)}(z^N)z^{-(N-1-i)} \tag{11}$$

Substituting (11) into (10), and using the Noble Identity [8, 1] to commute the filtering with the downsampling, we get

$$Y_n(z) = \sum_{i=0}^{N-1} H_i^{(n)}(z)\,(\downarrow N)\,X''(z)z^{-(N-1-i)}z^{(N-1)} \tag{12}$$

For $i = 0, 1, \cdots, N-1$

$$X_i(z) = (\downarrow N)\,X''(z)z^i \tag{13}$$

where $X_i(z)$ denotes the z-transform of the ith component of the input signal vector.

Thus, we have the following decomposition of each component of the output signal vector sequence $y_n(\cdot)$ in terms of conventional scalar filtering operations on each of the components of the input signal vector sequence:

$$Y_n(z) = \sum_{i=0}^{N-1} H_i^{(n)}(z)X_i(z) \tag{14}$$

The filtering can be expressed in the time domain as

$$y_n(m) = \sum_{i=0}^{N-1} \sum_{k=0}^{M-1} h_i^{(n)}(k) x_i(m-k) \qquad (15)$$

This filtering is illustrated in Fig. 1 where the filters $h_i^{(n)}$, $n = 0, 1, \cdots, N-1$ and $i = 0, 1, \cdots, N-1$ are the polyphase components of the nth row of the matrix $\tilde{\mathbf{\Gamma}}$ corresponding to the elements that multiply the ith component of the input signal vector in (4).

5. EFFICIENCY ANALYSIS

Since $\tilde{\mathbf{\Gamma}}$ is fixed, the polyphase components of its rows may be precomputed. The result is a set of conventional filters with scalar coefficients, which enables the components of the input signal vector sequence to be buffered and filtered independently of the other colors. Since the size of a row of $\tilde{\mathbf{\Gamma}}$ is $\frac{1}{N}$, the throughput is increased by a factor of N when the filtering is implemented in parallel. Because each filter in the parallel filterbank of Fig. 1 has M scalar coefficients, the rate at which these filters operate to deliver the same throughput is divided by a factor of N over the single processor implementation of (4).

Equation (4) performs MN^2 multiply accumulates in a sequential fashion to compute all components of the output signal vector. If we consider the filtering of an RGB image [6], then the parallel implementation would speed up the processing by a factor of three, although the total number of multiply-accumulates remains the same. We may utilize three low-bandwidth, low-cost processors instead of one high bandwidth processor to obtain the same performance at a lower cost [1], or exploit the instruction-level parallelism of a SIMD or VLIW processor such as the Intel Pentium MMX or Texas Instruments TMS320C6000, respectively. The efficient implementation computes the output N times as fast if all of the operations are performed at the same speed. The parallel implementation of Fig. 1 does not require shared circular buffers. Each component of the input vector sequence is put into a separate circular buffer on one of the N parallel processors. This allows for fast, low-overhead loop code.

6. EXTENSION TO MULTIDIMENSIONAL VECTOR-VALUED SIGNALS

The parallel implementation may easily be extended for to the filtering of multidimensional signals such as images. The multifilter operation on an L-dimensional N-vector valued signal is given by

$$\vec{\mathbf{y}}(\vec{\mathbf{m}}) = \sum_{\vec{\mathbf{k}} \in \mathcal{R}} \tilde{\mathbf{H}}(\vec{\mathbf{k}}) \vec{\mathbf{x}}(\vec{\mathbf{m}} - \vec{\mathbf{k}}) \qquad (16)$$

where $\vec{\mathbf{m}}$ and $\vec{\mathbf{k}}$ are L-dimensional vectors and $\mathcal{R}$ is the region of support of the multifilter.

The multifilter is equivalent to a block filter consisting of L samples per block. The filter mask is specified in blocks, and the filter moves from block to block instead of from sample by sample. At each block, the filtering given by (16) computes N linear combinations of all of the samples within the block mask. This filtering is illustrated in Fig. 2 for the case $L = 2$, $N = 3$ (RGB image), and the four matrix-coefficient multifilter ("error-filter") mask in [6]. Each component of the output N-vector is then recovered by filtering and downsampling the vector sequence rearranged into blocks (see Fig. 2) by using the scalar L-dimensional filter corresponding to the rows of the matrix $\tilde{\mathbf{\Gamma}}'$ by

$$\tilde{\mathbf{\Gamma}}' = \left[\tilde{\mathbf{H}}_0 \mid \tilde{\mathbf{H}}_1 \mid \ldots \mid \tilde{\mathbf{H}}_{|\mathcal{R}|-1} \right] \qquad (17)$$

where $\mid \mathcal{R} \mid$ is the cardinality of $\mathcal{R}$ and the $\tilde{\mathbf{H}}_i$ represent the coefficients of the multifilter within the support ordered arbitrarily. The downsampling matrix is given by $\Lambda = N\tilde{\mathbf{I}}_{N \times N}$. With this interpretation of block filtering in L dimensions and $(\downarrow \Lambda)$ replacing $(\downarrow N)$, we can use essentially the same arguments as in Section 4 to obtain the parallel form for the multifilter given by

$$y_n(\vec{\mathbf{m}}) = \sum_{i=0}^{N-1} \sum_{\vec{\mathbf{k}} \in \mathcal{R}} h_i^{(n)}(\vec{\mathbf{k}}) x_i(\vec{\mathbf{m}} - \vec{\mathbf{k}}) \qquad (18)$$

The speedup due to parallel implementation is N which is identical to the one-dimensional case.

7. CONCLUSION

This paper presents an efficient implementation of filters with matrix valued coefficients. For filters with M $N \times N$ coefficients, there are three possible implementations:

1. use the convolution definition on a single processor, which involves a sum of matrix-vector calculations: MN^2 multiply-accumulates
2. apply N scalar filters in parallel N times with different coefficients each time: M multiply accumulates per filter
3. apply N^2 scalar filters in parallel: M multiply accumulates per filter

We show that filtering on an N-vector valued sequence may be accomplished efficiently by using N^2 scalar valued filters, with N filters applied in parallel N times. We do not apply the N^2 filters in parallel because that would involve data duplication of the input stream and result in increased buffer size over the naive implementation. The parallel implementation however, involves synchronization of the parallel processors *after* each parallel filter has finished its filtering operation, if N parallel processors are used to implement the filtering.

8. REFERENCES

[1] G. Strang and T. Q. Nguyen, *Wavelets and Filter Banks*, Wellesley-Cambridge Press, 1996.

[2] G. C. Donovan, J. S. Geronimo, D. P. Hardin, and P. R. Massopust, "Construction of orthogonal wavelets using fractal interpolation functions", *SIAM Journal of Mathematical Analysis* vol. 27, no. 4, pp. 1158–1192, July 1996.

[3] G. Strang and V. Strela, "Short wavelets and matrix dilation equations", *IEEE Trans. on Signal Processing* vol. 43, no. 1, pp. 108–115, Jan. 1995.

[4] X.-G. Xia and B. W. Suter, "Multirate filter banks with block sampling", *IEEE Trans. on Signal Processing*, vol. 44, no. 3, pp. 484–496, March 1996.

[5] V. Strela, P. N. Heller, G. Strang, P. Topiwala and C. Heil, "The application of multiwavelet filterbanks to image processing", *IEEE Trans. on Image Processing* vol. 8, no. 4, pp. 548–563, Apr. 1999.

[6] L. Akarun, Y. Yardimci and A. E. Cetin, "Adaptive methods for dithering color images", *IEEE Trans. on Image Processing* vol. 6, no. 7, pp. 950–955. July 1997.

[7] P. Lapsley, J. Bier, A. Shoham and E. A. Lee, *DSP Processor Fundamentals*, Berkeley Design Technology, Inc., 1996

[8] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*, Prentice-Hall, Inc., 1993.

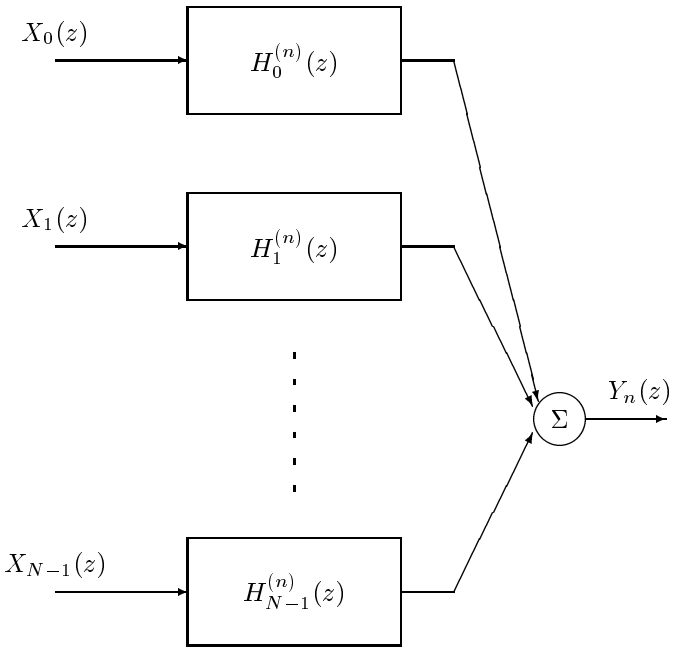

Figure 1: A parallel implementation of a multifilter which computes the nth component of the output signal N-vector sequence. N^2 filters are required for the computation of all of the components of the output vector sequence.

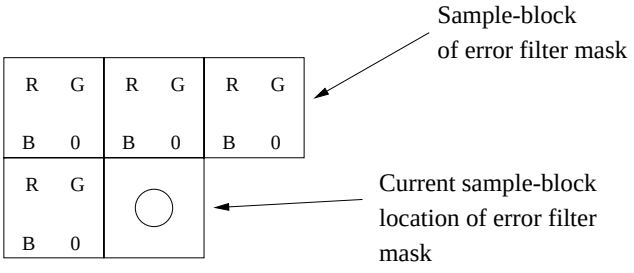

Figure 2: Example multifilter operating on an RGB image expressed as sample blocks. Three linear combinations of all samples in the mask are computed at the current location (denoted by the circle) to compute the red, green and blue samples of the multifilter output. The filter moves in a raster scan of the blocks.