Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2007, Article ID 91741, 10 pages
doi:10.1155/2007/91741

Research Article

On the Vectorization of FIR Filterbanks

Jayme Garcia Arnal Barbedo and Amauri Lopes

Department of Communications, FEEC, State University of Campinas (UNICAMP), P.O. Box 6101, 13083-970 Campinas, SP, Brazil

Received 20 October 2005; Revised 23 May 2006; Accepted 22 June 2006

Recommended by Roger Woods

This paper presents a vectorization technique to implement FIR filterbanks. The word vectorization, in the context of this work, refers to a strategy in which all iterative operations are replaced by equivalent vector and matrix operations. This approach allows that the increasing parallelism of the most recent computer processors and systems be properly explored. The vectorization techniques are applied to two kinds of FIR filterbanks (conventional and recursive), and are presented in such a way that they can be easily extended to any kind of FIR filterbanks. The vectorization approach is compared to other kinds of implementation that do not explore the parallelism, and also to a previous FIR filter vectorization approach. The tests were performed in Matlab and *C*, in order to explore different aspects of the proposed technique.

1. INTRODUCTION

Since its beginning, the fast Fourier transform (FFT) has been one of the most popular techniques for time-frequency decomposition. The arising of faster FFT algorithms [1, 2] caused an even more pronounced supremacy. However, the properties of the time-frequency decomposition performed by FFT do not match with the requirements of certain applications, especially when good temporal and spectral resolutions are demanded at the same time. In those cases, other techniques must be considered. One of such alternatives is the finite impulse response (FIR) filterbank.

Although filterbanks have several advantages over FFT [3], the high computational complexity associated to them often implies their replacement by FFT, even with sacrifice of the temporal or spectral resolution. In this context, this paper aims to provide a fast and effective implementation of FIR filterbanks by using vectorization techniques which are able to efficiently explore the increasing parallelism of modern microprocessors, vector processors, and supercomputers. Moreover, it is intended that the information presented in this paper inspire the development of new efficient codes in different areas of digital signal processing.

The word vectorization is often associated to the high-performance computational field, by using supercomputers with great number of parallel processors or vector processors highly specialized to deal with vector and matrix operations [4–7]. Nevertheless, the microprocessors used in personal computers have gradually incorporated parallel computational capabilities in order to improve their performance. In the context of this work, the vectorization is associated to the substitution of iterative segments of a code by vector and matrix operations.

All tests to assess the performance of the vectorization techniques proposed here were carried out in a computer with conventional processor. Codes written in *C* were used whenever the main goal was to compare the proposed approach with previous techniques, which are often implemented in *C*. On the other hand, codes written in Matlab were preferred when the main goal was showing the relative difference between the runtimes of vectorized and nonvectorized codes. In this context, Matlab shows several desirable characteristics, like easier implementation and better visualization of the vectorization effects, since purely vector codes written in this tool can be much faster than their loop-based versions. This occurs because Matlab uses the processor's registers to store the vectors instead of sending and recovering them from memory, saving lots of time and making the execution much faster. In other words, it automatically uses the parallelism capability of the processor.

The vectorizing techniques to be presented next are useful not only in cases where the implementations are carried out in Matlab or *C*, but also in situations where other general purpose programming languages are used together

with vectorizing compilers. In this last case, the information present in the paper can make the construction of vectorizable loops quite straightforward. In the case of Matlab, the procedure is even simpler, since the equations must be implemented exactly as presented in the following Sections.

Finally, it is important to underline that the following sections include some optimization techniques that are not directly related to vectorization. The most important of such techniques is the division of the signals into frames, which aims to reduce memory requirements. This procedure is particularly effective when long signals are considered, because the memory requirements are no longer determined by the length of the entire signal, but by the length of each frame. The association of the signal division with vectorization techniques led to good results, as presented in Section 5.

The paper is divided as follows. Section 2 presents a brief discussion about related works; Section 3 explores the vectorization applied to decimation finite impulse response filterbanks; Section 4 presents a vectorization technique applied to a specific example of a recursive FIR filterbank, which combines characteristics from both FIR and IIR filterbanks, as well as some particular features; Section 5 describes the tests and corresponding results; finally, Section 6 presents some conclusions.

2. RELATED WORKS

The optimization of filters and filterbanks computational performance is not a new task. The efforts to find efficient implementations have begun practically together with the digital signal processing field itself, and lots of techniques have been proposed so far. This section presents some of the most important of those works. The first part of the section presents some general proposals, while the second part is dedicated to works dealing with vectorization.

An interesting early work dealing with the efficient implementation of filterbanks was [8]. The author presented an optimized implementation of a decimation filterbank used in speech recognition applications. The techniques used to reduce the computational complexity were dithering and the Winograd Fourier transform algorithm.

In [9], the authors use genetic algorithms to design low complexity digital FIR filters. The proposed method also uses a primitive operator directed graph implementation to reduce the computational complexity.

A combination of minimum-adder canonic signed digit (CSD) multiplier blocks with a technique that trades adders for delays is used in [10] to reduce the hardware requirements for fixed coefficient FIR filters.

In [11], the authors present a public domain Matlab program that generates optimized VHDL descriptions of filter implementations, using CSD or DM (Dempster and Macleod) techniques.

An optimized structure for decimation filterbanks to be used in mobile systems is the focus of the techniques proposed in [12]. The final goal is a hardware efficient VLSI implementation.

The optimization of nearly perfect reconstruction FIR cosine-modulated filterbanks is presented in [13]. The implementation is based on a new expression for the analysis bank.

The optimization procedures of the works presented next are all based on vectorization techniques.

An important early work dealing specifically with vectorization was [14]. The authors present a number of vectorization methods applied to the implementation of digital filters in pipelined vector processors.

Reference [15] deals with the subject of high sampling rate realizations for transversal adaptive filters. A parallel algorithm is mapped onto a linear array of highly pipelined processing modules, resulting in a system able to efficiently implement transversal adaptive filters.

In [16], the authors present a tool that eases the conversion of conventional DSP programs into vector operations using simple vector units.

An efficient implementation of recursive digital filters into vector SIMD DSP architectures is presented in [17]. Vector DSPs are also the focus of references [18, 19].

Some ideas present in previous works inspired part of the strategy presented in this paper, but the general approach of the method is quite different from its predecessors, as will be seen in next sections.

3. VECTOR IMPLEMENTATION OF DECIMATION FIR FILTERBANK

There are several situations that require some kind of signal decimation. It is common that the decimation be associated to a filtering process. In general, both procedures can be combined in such a way that computational resources are saved. This situation has motivated the use of a decimation FIR filterbank instead of a regular one, making the techniques presented here more general. The procedure for nondecimation FIR filterbanks can be obtained by simply making the decimation factor presented in (1) equal to one.

In this section, a signal $x(n)$, $1 \le n \le N_s$, to be filtered by a decimation FIR filterbank, is considered. The kth filter, $1 \le k \le K$, has coefficients b_{ki}, $1 \le i \le C_{fk}$. The corresponding signal at the output of the kth filter is

$$y_k(n) = \sum_{i=1}^{C_{fk}} b_{ki} \cdot x(n-i), \quad n = D, 2D, 3D, \ldots, \qquad (1)$$

where D is the desired decimation factor.

The vectorial procedure to implement the filtering process has three main goals: (1) the FIR filtering convolutions must be carried out using multiplication of matrices instead of loops; (2) all filters in the filterbank must be applied at once; (3) the decimation must be performed during the filtering, and not after, in such a way that the calculations are done only for those output samples to be considered after the decimation. This particular filtering process was chosen because it contains a number of procedures commonly used in the implementation of filters. In this way, the techniques can be easily extended both to simpler and more complex implementations.

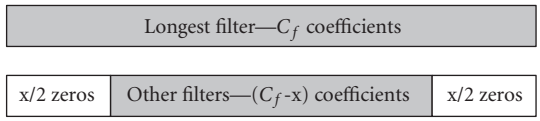

Figure 1: Filter length equalization.

The strategy to be presented can be divided into six steps:

(1) the coefficient vectors of the filters are prepared to be submitted to the next processing in step (2);
(2) the coefficient vectors are grouped into a single matrix, the coefficient matrix;
(3) the signal to be filtered is divided into frames;
(4) each frame is split into subframes, which are grouped into a matrix, the frame matrix;
(5) each frame matrix is multiplied by the coefficient matrix, producing the corresponding convolved matrix, that is, the matrix composed of the corresponding filterbank output;
(6) the convolved matrices are concatenated, generating the final time-frequency decomposition of the signal.

As can be seen, the first two steps are related to the preprocessing of the filters, the next two prepare the signal for filtering and the last two perform the filtering. The details of the steps are presented next.

3.1. Preparing the filters for the vector processing

Firstly, the number of coefficients of each filter must be adjusted to match the number of coefficients of the filter with longest impulse response. Moreover, the coefficient vectors must be aligned in such a way that the center coefficients match the same position along the vectors. This procedure is necessary to prepare the coefficients for the convolution to be performed in following steps.

This adjustment is done by adding zeros at the beginning and at the end of each coefficient vector, as shown in Figure 1. If the difference between the number of coefficients is odd, an extra null coefficient must be located at the beginning of the vector.

After the length adjustment, each sequence of coefficients must be reversed, meaning that the last coefficient becomes the first, the penultimate becomes the second, and so on.

Finally, the reversed coefficient vectors are grouped into a single K-by-C_f matrix, here named $\mathbf{C_k}$, where C_f is the length of the longest impulse response. Note that the kth row of matrix $\mathbf{C_k}$ is the reversed coefficient vector for the kth filter.

3.2. Division of signal

The signal must be divided into frames aiming to reduce the amount of data to be stored in the memory at a time. This procedure has practically no impact on the number of mathematical operations, but makes storing, accessing, and retrieving the data much faster, as can be seen in the results presented in Section 5. The designer must choose a frame size adequate to the available computational resources and the characteristics of his project. Figure 2 illustrates this division.

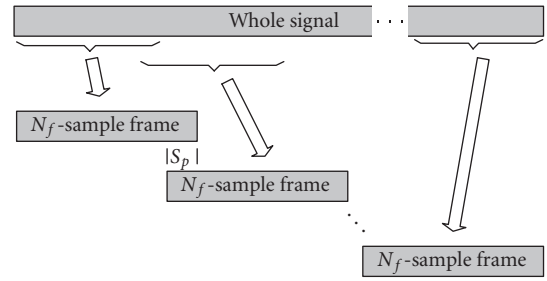

Figure 2: Division of the signal into frames.

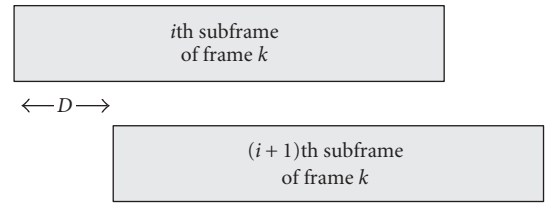

Figure 3: Delay between consecutive frames.

In Figure 2, N_f is the length of the frames and S_p is the superposition between the frames. This superposition is necessary to assure that the filtering will be correctly performed, as will be seen in Section 3.3.

3.3. Subdivision of the frames

Each frame is divided into subframes with C_f samples. Each subframe corresponds to the ensemble of samples necessary to produce an output sample. Also, the beginning of a subframe is D samples after the beginning of the last subframe, as shown in Figure 3, in order to take into account the desired decimation factor D.

Figure 4 shows that the last subframe of a frame will not necessarily exactly fit the end of the respective frame. In this case, a number of samples will remain unprocessed (a in Figure 4). Those samples must be considered in the next frame. As a consequence, the beginning of the next frame must be at the sample located at D samples after the beginning of the last subframe. This arrangement justifies the superposition between consecutive frames mentioned in Section 3.2.

The superposition between frames is

$$S_p = N_f - D \cdot (1 + R), \tag{2}$$

where $R = \lfloor (N_f - C_f)/D \rfloor$.

After this division, the subframes of the ith frame are concatenated into an R-by-C_f matrix, named $\mathbf{X}(i)$, as shown in Figure 5. This matrix allows that the filter coefficients be

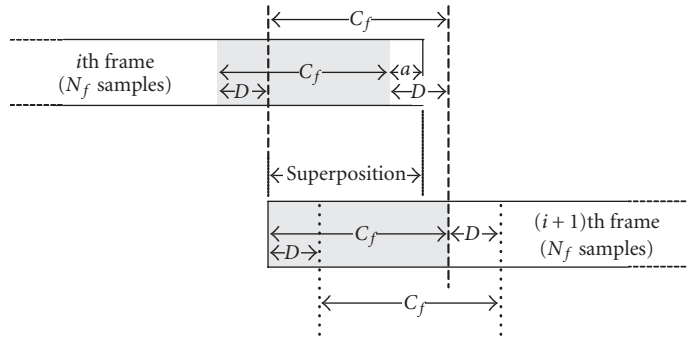

Figure 4: Superposition between the frames.

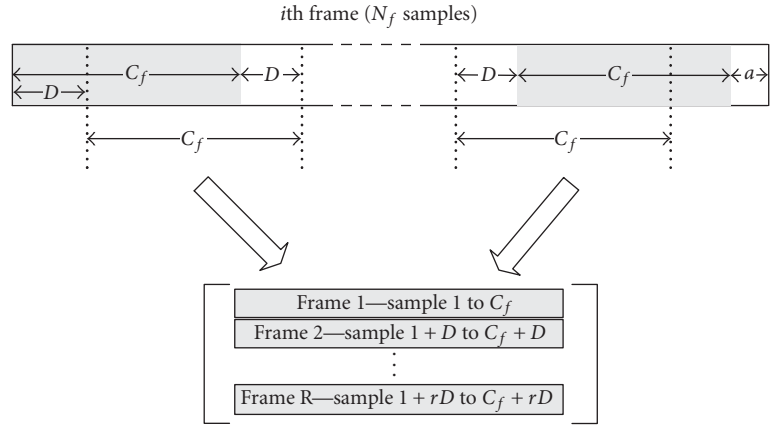

Figure 5: Concatenation of subframes into a matrix.

applied matricially to the whole signal, in such a way that all K filters are applied at a time.

3.4. Matrix filtering

Next, the matrix filtering is performed according to

$$\mathbf{C}_{K\times C_f} \cdot \mathbf{X}^{\mathbf{T}}_{C_f\times R}(i) = \mathbf{F}_{K\times R}(i), \tag{3}$$

where $\mathbf{X}^{\mathbf{T}}$ denotes the transposed of $\mathbf{X}$. The rows of matrix $\mathbf{F}(i)$ are the signals at the output of the filters, corresponding to the ith frame at the input. This procedure is repeated for all frames (index i in (3)).

3.5. Concatenation of results

The matrices $F(i)$ are concatenated into a single matrix G according to (4), where M is the number of frames. The rows of matrix $\mathbf{G}$ are the signals at the output of the filterbank, corresponding to the entire signal $x(n)$ at the input,

$$\mathbf{G} = \Big[\mathbf{F}(1) \;\; \mathbf{F}(2) \;\; \cdots \;\; \mathbf{F}(M)\Big]. \tag{4}$$

Note that the procedure described here can be applied to signals of any length. Moreover, the procedure can be applied even if the length is unknown. In any circumstance there will be an output delay of one frame or more.

4. VECTOR IMPLEMENTATION OF DECIMATION RECURSIVE FIR FILTERBANK

This section presents vectorization techniques for a specific FIR filterbank implemented in a recursive way. This recursion is obtained by means of a pole added to the system function; a zero, at the same position, cancels the pole. This particular form is motivated by a proposal presented in [3] for a bandpass filterbank.

4.1. Description of the filterbank

The kth filter of the bank, $1 \le k \le K$, is described by the difference equation

$$y_k(n) = \sum_{m=0}^{D-1} \left[a_{km} \cdot x(n-m) - a^*_{km} \cdot x(n-m-1+D+C_{fk})\right] + b_k \cdot y(n-D), \tag{5}$$

where

$$\begin{gathered} n = 1, D+1, 2D+1, \ldots, \\ a_{km} = e^{[j\cdot(M-(C_{fk}+D-1/2))\cdot\Omega_{Ck}]}, \\ b_k = e^{j\cdot D\cdot\Omega_{Ck}} \\ \text{for } n \leq 0 \longrightarrow y(n) = 0, \quad x(n) = 0, \end{gathered} \tag{6}$$

D is the decimation factor, which must be smaller than the order C_{jk} of the filters. Note that the recursive part of the filters corresponds to the feedback of a single output sample.

The nonrecursive part involves two terms. Each of those terms uses only D samples of the signal $x(n)$ to produce an output sample. This is a special situation that demands additional vectorization procedures because the application of the procedures presented in Section 3 would lead to a sparse coefficient matrix, with zero elements in the positions that do not play a role in the filtering. This sparse matrix would demand useless computational effort due to multiplications by zero.

Therefore, it is necessary to create a procedure to calculate the nonrecursive part of (5).

4.2. Implementation of the nonrecursive part

This proposal follows the same general strategy described in Section 3. Then, the first task is the division of the signal $x(n)$ into frames with N_f samples in order to reduce memory requirements.

Next, each frame is divided into subframes. However, the frame division must be performed carefully, since some questions must be considered: (1) the length C_{fk} of the filters can vary considerably, depending on the passband width of each filter; (2) the relative position of the filter coefficients and the signal must be adjusted in order to keep the filtered versions of the signal aligned. This implies that the center coefficient of each filter must be aligned with the same signal sample; (3) as can be seen in (5), the first term of the nonrecursive part uses the samples $x(n), x(n-1), \ldots, x(n-D+1)$, while the second term uses the samples $x(n-1+D+C_{fk}), x(n-2+D+C_{fk}), \ldots, x(n+C_{fk})$. Those samples are located at the opposite extremes of a segment of a signal with length of $C_{fk}+D$ samples.

The frame division proposed here creates subframes with D samples (equal to the decimation factor). This is because each term of the nonrecursive part in (5) uses only D samples of $x(n)$ to produce an output sample.

The frame division is illustrated in Figure 6, where the decimation factor is $D = 8$ and the highest filter order is $C_f = 60$. A 40th-order filter is also shown in the example. Each frame is, therefore, divided into 8-sample segments. The following procedures must be carried out.

(i) In the case of the highest-order filter (Figure 6(a)), the first $D = 8$ coefficients are applied to the first eight samples of the signal (situation 1). Unless the order of the filter is a multiple of eight, the last eight coefficients of the filter will not be applied to the correct samples, as in the example. To align the last eight coefficients of the highest-order filter with the correct samples of the signal, a new splitting must be applied. In the example, the new division must begin at the 5th sample of the signal, ignoring the first four samples; thus, a correct alignment is accomplished (situation 2).

(ii) The situation shown in Figure 6(b) refers to the 40th-order filter, whose center must be aligned to the center of the highest-order filter. In this case, the eight first coefficients of the 40th-order filter will not be applied to the eight first samples of the signal, and in most cases, the samples to be weighted by the coefficients will be located in different segments of the signal (situation 3). To correct this mismatch, the new splitting must begin at the 3rd sample, ignoring the first two samples (situation 4). As this filter has an order that is a multiple of the decimation factor, this alignment is also appropriate for the last coefficients. If this was not true, a new splitting must be carried out.

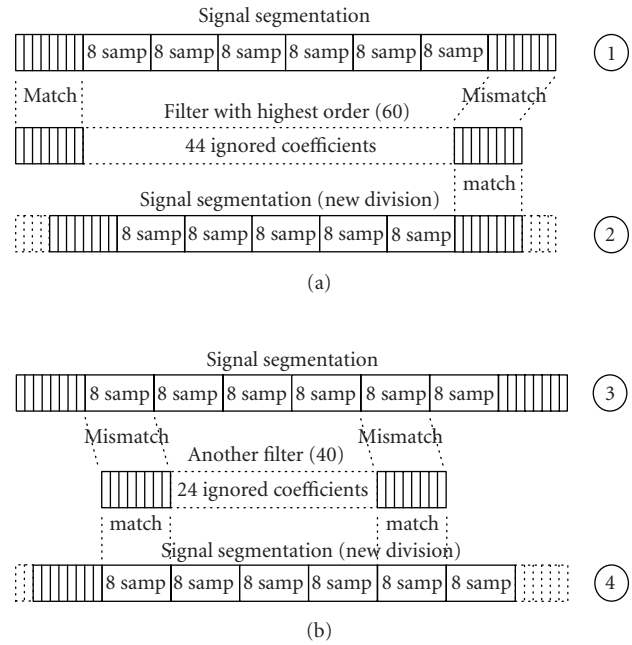

Figure 6: Strategy to adjust the filter coefficients.

The same procedure must be applied to all other lower order filters of the bank.

As can be seen, depending on the number of filters, the signal must be split as many times as the decimation factor. This situation increases the amount of data to be stored, justifying the first division of the signal into frames. However, despite the frame division, the additional processing demanded by the splitting can be a problem if the decimation factor is high. One possible solution, which was adopted here, is to force the filter orders to be a multiple of some number. For instance, in a case where $D = 32$ and the order of the filters is forced to be a multiple of 8, there will be at most 8 possible different alignments, as illustrated in Figure 7.

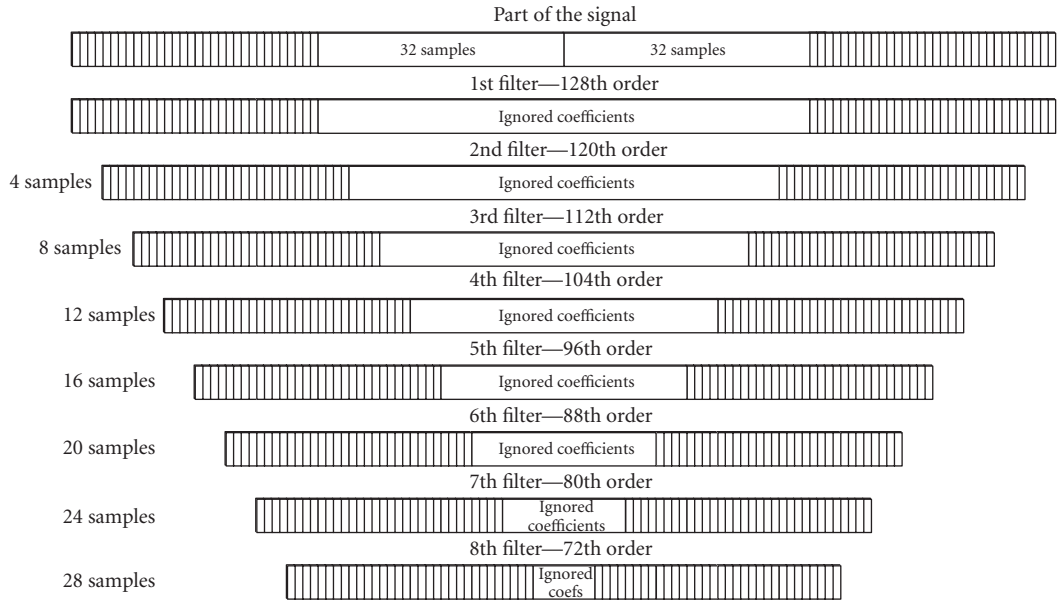

Figure 7: Example of filterbank design.

The number of samples shown in the left of Figure 7 indicates the number of samples to be discarded from the signal for each case. In the case of Figure 7, the number of splits to be applied to the signal is determined by half the difference between the lengths of two consecutive filters. This is because the filters must have the center coefficients aligned and the difference between their lengths will be equally distributed between both extremities. Therefore, the number of splits for this example is $32/4 = 8$.

This is the maximum number of splits required when the filter orders are multiples of a number H. This maximum occurs when there are as many filter orders as the multiples of H inside the range between the lowest to highest orders. Therefore, the maximum number S of splits for the proposed procedure is

$$S = \frac{2 \cdot D}{H}. \tag{7}$$

Note that increasing the value of H reduces the filter design flexibility. The designer must determine the compromise between flexibility and memory requirements based on the characteristics of the project.

Finally, it is important to emphasize that all possible signal splits are performed and stored before applying the filters to the signal. This procedure increases the amount of data to be stored, but saves lots of computational resources, since each split is performed only once.

4.3. Performing the summation

As described before, all split versions of a frame will be generated before the filtering procedure and will be stored. Additionally, the filters will be grouped according to the corresponding split version required. Hence, the number of groups will be equal to the number of splits applied to the signal. The expression to determine in which group a given filter must be is given by

$$s = \frac{(C_{fk} \bmod 2D) + 2D}{H}, \tag{8}$$

where "mod $2D$" is the module 2D operation.

Using the example of Figure 7, the first filter pertains to group 8, the second to group 7, and so on, until the eighth filter, which pertains to group 1. The possible following filters would repeat such classifications, being grouped accordingly. In this case, the 64th-order filter would be grouped together with the 128th-order filter, the 56th with the 120th, and so on.

In order to present the proposed concatenation of the filter coefficients, note that the expression inside the summation in (5) is divided into two terms: the first one makes use of the first D coefficients of the filters, here called $f_k(i)$; the second one makes use of the last D coefficients of the filters, here called $g_k(i)$.

The coefficients $f_k(i)$ and $g_k(i)$ of the filters pertaining to a certain group are arranged into matrices $\mathbf{F_s}$ and $\mathbf{G_s}$, respectively. The index s varies from 1 to S, and indicates the filter groups. The rows of matrix $\mathbf{F_s}$ are the coefficients $f_k(i)$ of those filters that pertain to group s. In the same way, the rows of matrix $\mathbf{G_s}$ are the coefficients $g_k(i)$ of the filters that pertain to group s . Therefore, matrices $\mathbf{F}_s$ and $\mathbf{G}_s$ have D columns and a number of rows equal to the number of filters that pertain to group s.

The subframes corresponding to the split group s are concatenated as the columns of a matrix $\mathbf{X_s}$ with dimensions $D \times \lfloor N_f/D \rfloor$. After that, the summation of each term in (5)

is calculated by

$$\begin{aligned} \mathbf{P}_s &= \mathbf{F}_s \cdot \mathbf{X}_s, \\ \mathbf{Q}_s &= \mathbf{G}_s \cdot \mathbf{X}_s. \end{aligned} \tag{9}$$

At this point, matrices $\mathbf{P}_s$ and $\mathbf{Q}_s$, for all values of s, contain a number of patterns resulting from the filtering process, but they are not correctly ordered, because the previous grouping of filters does not respect the original sequence of filters. Therefore, the matrices $\mathbf{P}_s$ and $\mathbf{Q}_s$ must not only be concatenated, but the sequence of filters must be restored. This procedure is indicated by the operator $O(\cdot)$ in the following equations:

$$\begin{aligned} \mathbf{P} &= O([\mathbf{P}_s]), \\ \mathbf{Q} &= O([\mathbf{Q}_s]). \end{aligned} \tag{10}$$

Finally, the matrices $\mathbf{P}$ and $\mathbf{Q}$ are combined according to (5) as

$$\mathbf{C} = \mathbf{P} - \mathbf{Q}. \tag{11}$$

This procedure completes the nonrecursive part of (5) for a frame.

4.4. Implementation of the recursive part

The factor b_k that multiplies $\mathbf{y}_k$ in the last part of (5) is a constant for each filter. Considering that the summation of the nonrecursive part has already been determined, (5) can be rewritten as

$$\mathbf{y}_k(i) = \mathbf{c}_k(i) + b_k \cdot \mathbf{y}_k(i-1). \tag{12}$$

In (12), i varies from 1 to L (length of the frames at the output of the filters) and $\mathbf{c}_k(i)$ is the summation vector for the kth filter and ith sample, extracted from the matrix $\mathbf{C}$. Expanding (12) results in

$$\begin{aligned} \mathbf{y}_k(1) &= \mathbf{c}_k(1), \\ \mathbf{y}_k(2) &= \mathbf{c}_k(2) + b_k \cdot \mathbf{c}_k(1), \\ \mathbf{y}_k(3) &= \mathbf{c}_k(3) + b_k \cdot \mathbf{c}_k(2) + b_k^2 \cdot \mathbf{c}_k(1), \\ &\vdots \\ \mathbf{y}_k(L) &= \mathbf{c}_k(L) + b_k \cdot \mathbf{c}_k(L-1) + \cdots + b_k^{L-1} \cdot \mathbf{c}_k(1). \end{aligned} \tag{13}$$

Equation (13) is equivalent to a convolution between the vectors $\mathbf{c}_k(i)$ and the vectors $[1 \;\; b_k \;\; b_k^2 \;\; \cdots \;\; b_k^{L-2} \;\; b_k^{L-1}]$. Both sets of vectors can be grouped into matrices in such a way that (13) can be written as

$$\mathbf{Y} = \mathbf{C} \otimes \mathbf{B}, \tag{14}$$

where $\otimes$ is the convolution between the corresponding lines of matrices $\mathbf{C}$ and $\mathbf{B}$. Performing this convolution in time-domain implies a high computational cost. Thus, the best alternative is to perform the convolution in the frequency domain, as given by

$$\mathbf{D} = \Im\{[\mathbf{ZB}]\}, \qquad \mathbf{E} = \Im\{[\mathbf{ZC}]\}, \tag{15}$$

$$\mathbf{Y} = \Im^{-1}\{\mathbf{D} \cdot \mathbf{E}\}. \tag{16}$$

In (15) and (16), $\Im$ indicates the FFT, $\Im^{-1}$ the inverse FFT, $\mathbf{Z}$ is an all-zero matrix with the same dimensions of matrices $\mathbf{B}$ and $\mathbf{C}$, and the multiplication in (16) is scalar, meaning that an element of one matrix will multiply only its correspondent in the other one. The matrix $\mathbf{Z}$ is concatenated with the other ones in order to change the convolution from circular to linear.

It is important to note that matrix $\mathbf{B}$ depends only on the filters. Therefore, matrix $\mathbf{B}$ is known a priori and its FFT can be calculated and stored before the filtering. This procedure can save lots of computation, and the only shortcoming is the physical memory resources needed. Nevertheless, the size of the matrix is almost always insignificant compared to the computational resources available in most systems.

The matrix $\mathbf{Y}$ resulting from the process corresponds to the time-domain output of the filterbank.

4.5. Considerations on the IIR filterbanks vectorization

Due to the intrinsic recursive nature of IIR filters, only the nonrecursive part of this kind of structure can be directly vectorized using the strategies described in Section 3. However, some particular implementations can benefit from the techniques described in this section. The degree of vectorization that can be reached in such cases will depend on the characteristics of the project and also on the ability of the designer in identifying possible vectorizable code segments.

5. TESTS AND RESULTS

5.1. Description of the filterbank used in the tests

The filterbank used in the tests is an approximate model to the frequency separation performed by the human ear, which consists of 40 filters [20–22]. The passbands have different widths in Hertz, but are equally spaced and have a constant bandwidth when measured in a perceptual scale. The center frequencies vary from 50 Hz to 18 kHz. The envelopes of the impulse responses have a $\cos^2$ shape. The filter coefficients are given by [22]

$$h(k,n) = \begin{cases} \dfrac{4}{N[k]} \cdot \operatorname{sen}^2\left(\dfrac{\pi \cdot n}{N[k]}\right) \\ \quad \cdot \cos\left(2\pi \cdot f[k] \cdot \left(n - \dfrac{N[k]}{2}\right) \cdot T\right), & 0 \le n < N[k], \\ 0; & \text{otherwise}, \end{cases} \tag{17}$$

where k is the filter index, n is the time sample index, T is the time between two samples, $N[k]$ is the length of the impulse response, and $f(k)$ is the center frequency of the kth band in Hertz. During the filtering, the signals are decimated by a factor of 32. This filterbank was implemented using both strategies presented in Sections 3 (FIR filterbank) and 4 (recursive FIR filterbank).

5.2. Results

The tests were designed to compare the performance of the proposed strategy with nonvectorized codes, and also with another vectorization strategy found in the literature. The results achieved for conventional and recursive FIR filterbanks are presented separately.

5.2.1. FIR filterbank

Six different implementations were tested for the filterbank, as described in the following.

(1) *All-sample approach using loops*: in this implementation, the filtering is done using loops; additionally, the decimation is done after the signal has been filtered.

(2) *Selected-sample approach using loops*: this version also uses loops, but calculates only those samples to be considered after the decimation.

(3) *Quantization of the filter coefficients*: there are some applications for which the quality of the filtered signal remains satisfactory if the filter coefficients are quantized; this procedure reduces drastically the number of multiplications, since it is possible to group and sum samples to be submitted to a same quantized coefficient before performing the multiplication; decimation is performed during the filtering, as described in the second approach; this strategy also uses loops.

(4) *Frequency-domain multiplication*: the signals and filter coefficients are submitted to a fast Fourier transform (FFT), the resulting patterns are multiplied and the inverse FFT is calculated; the decimation is performed after the filtering procedure.

(5) *Overlap-and-save approach*: it is quite similar to the previous approach, but it reduces the amount of memory required at a time by dividing the signal into frames and combining the filtered segments according to the overlap-and-save methodology [23]; decimation is also performed after the filtering procedure.

(6) *Vectorized approach*: it uses the procedure described in Section 3.

Two audio excerpts sampled at 48 kHz and with durations of 2 and 20 seconds were used in the tests. The experiments were performed in a microcomputer with processor AMD Athlon 2000+, 512 MB of RAM, and Microsoft Windows XP as operational system. All tests and implementations were performed using Matlab 6.5. The results for each approach are shown in Table 1, and the comments are presented in the following.

It is important to highlight that the computation time required by each algorithm was used as parameter of comparison, instead of the number of flops. This is because the number of flops is related to the number of operations, but the techniques proposed here were developed having in mind not only the reduction of the number of operations, but also the reduction of memory requirements. Therefore, techniques that do not result in fewer operations, but reduce the time needed to access memory, as the division of the signals into frames, can be properly considered and assessed.

Table 1: Results for the FIR filterbank.

Approach	Time required 2 seconds signal	Time required 20 seconds signal	*RI*
1	441.19 s	14.563.5 s	0.303
2	8.07 s	96.9 s	0.833
3	26.01 s	945.3 s	0.275
4	6.70 s	53.7 s	1.247
5	3.73 s	50.4 s	0.741
6	0.99 s	9.93 s	0.997

Another factor that has been considered in the comparison of the approaches is the index *RI* given by

$$RI = \frac{t_1}{t_2} \cdot \frac{d_2}{d_1}, \tag{18}$$

where t_1 and t_2 are the time spent to filter the first and the second signals, respectively, and d_1 and d_2 are the durations of first and second signals. This index indicates how the computation time varies with the length of the signal:

(i) if $RI = 1$, the time required will vary linearly with the length of the signals;
(ii) if $RI < 1$, the time spent will raise exponentially as the length of the signal is increased;
(iii) if $RI > 1$, the time will raise logarithmically as the length of the signal is increased.

High values of *RI* indicate good computational performance for longer signals. It is desirable that *RI* be at least 0.95.

The following remarks are drawn from Table 1.

(i) Approach 1 is the worst option, due to the excessive number of multiplications and the large amount of data to be stored and retrieved from memory during the process. The *RI* index indicates that the required computation time increases exponentially with the length of the signal, which is mostly due to the huge amount of memory required when the entire signal is considered at once.

(ii) The number of calculations for approach 2 is 32 times smaller than approach 1. Moreover, fewer samples are being considered. As a consequence, the memory resources are less stressed. However, although a lot of time has been saved, the overall time spent is still too expensive. The *RI* indicates that this approach is not appropriate to long signals, essentially due to the same reasons pointed out for approach 1.

(iii) The performance of approach 3 is very disappointing, because it was expected that the great reduction in the number of multiplications would improve the performance of the filtering. However, this approach requires that a large amount of data be continuously stored and retrieved from memory, making the process slower. The *RI* value does not recommend the use of this method for long signals.

(iv) Approach 4 was inefficient due to the large amount of data to be stored in the memory. *RI* is high, but its use only becomes advantageous for very long signals. However, in such cases the memory required can exceed the computational resources.

(v) Technique 5 presents better results than the previous ones, but its execution is still too slow. This is due to the

Table 2: Results for the recursive FIR filterbank.

Approach	Time demanded (s)
1	592.5
2	307.3
3	11.8

impossibility to perform the decimation directly during the calculation of the inverse FFT, yielding lots of unnecessary calculations. Nevertheless, fixing this problem would not be enough to make its performance superior to the vectorized approach. The *RI* is low.

(vi) As can be seen, the proposed technique (approach 6) is the fastest, confirming the effectiveness of such a strategy. Additionally, the high *RI* makes it appropriate for longer signals. In order to test the effect of splitting the signal into frames, approach 6 was also tested with the entire signal at once. This version spent, in average, twice the time required using the frame division, confirming the effectiveness of this action.

The implementation of the filterbank using approach 6 was also written in *C*. This version was compared with an implementation based on the VIOL (vectorizing inner and outer loops) approach presented in [19]. The proposed strategy is almost 2.5 times faster than the VIOL-based implementation. This means that the strategy not only provides a significant speedup over nonvectorized codes, but also presents a good performance compared with other FIR filter vectorization approaches.

5.2.2. Recursive FIR filterbank

The signal used here is the same as the 20-second excerpt used in the tests of the FIR filterbank (see Section 5.2.1). The specifications of the filterbank used here are also the same as that used in Section 5.2.1. The results for each approach are shown in Table 2, and the comments are presented in the following.

In approach 1, the filtering was implemented using for-loops instead of a vector-based approach, and the signal was not divided into frames. As can be seen, the results were very poor, since the parallelism of the processor was not explored at all. Furthermore, the time demanded increases exponentially with the length of the signal.

Approach 2 follows the same strategy of the first one, but here the memory requirements are reduced by dividing the signal into 96.000 sample frames. As a result, the time spent dropped nearly 50%, and this reduction tends to increase as longer signals are considered. Additionally, the time demanded increases almost linearly with the length of the signal. However, this strategy is still too slow.

Approach 3 is the one presented in Section 4. The program has run 26 times faster than the code implemented using the second approach, and its performance varies practically linearly with the length of the signal. These remarks support the theoretical advantages of vectorization.

This last approach was also tested using a *C* code. In this case, the proposed strategy was 3.2 times faster than the VIOL-based implementation. This result is even better than that one achieved for the regular FIR filterbank, confirming the effectiveness of the vectorization approaches of FIR filterbanks proposed in this paper.

6. CONCLUSION

A vectorized implementation of FIR filters, which is able to explore the growing parallelism present in modern computer processors, has been proposed. The technique has been presented in a generalized form, in such a way it can be extended to a large number of different FIR filter architectures.

The performance of the proposed strategy was assessed using codes written in both Matlab and *C*, and the results were compared with nonvectorized codes and also with a previous approach. In all cases, the proposed technique has provided significant speedup.

ACKNOWLEDGMENT

Special thanks are extended to FAPESP for supporting this work under Grants 01/04144-0 and 04/08281-0.

REFERENCES

[1] A. Edelman, P. McCorquodale, and S. Toledo, "The future fast Fourier transform?" *SIAM Journal on Scientific Computing*, vol. 20, no. 3, pp. 1094–1114, 1999.

[2] M. Frigo and S. G. Johnson, "FFTW: an adaptive software architecture for the FFT," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '98)*, vol. 3, pp. 1381–1384, Seattle, Wash, USA, May 1998.

[3] T. V. Thiede, *Perceptual audio quality assessment using a nonlinear filter bank*, Ph.D. thesis, Technical University of Berlin, Berlin, Germany, 1999.

[4] M. Weinhardt and W. Luk, "Pipeline vectorization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 2, pp. 234–248, 2001.

[5] T. Fahringer and B. Scholz, "A unified symbolic evaluation framework for parallelizing compilers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 11, pp. 1105–1125, 2000.

[6] W. Blume, R. Eigenmann, K. Faigin, et al., "Polaris: the next generation in parallelizing compilers," in *Proceedings of the 7th International Workshop in Languages and Compilers for Parallel Computing (LCPC '94)*, pp. 10.1–10.18, Ithaca, NY, USA, August 1994.

[7] H. Zima and B. Chapman, *Supercompilers for Parallel and Vector Computers*, Addison-Wesley, New York, NY, USA, 1990.

[8] H. F. Silverman, "A high-quality digital filterbank for speech recognition which runs in real time on a standard microprocessor," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 34, no. 5, pp. 1064–1073, 1986.

[9] D. W. Redmill and D. R. Bull, "Design of low complexity FIR filters using genetic algorithms and directed graphs," in *Proceedings of the 2nd International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pp. 168–173, Glasgow, UK, September 1997.

[10] M. A. Soderstrand, L. G. Johnson, H. Arichanthiran, M. D. Hoque, and R. Elangovan, "Reducing hardware requirement in FIR filter design," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '00)*, vol. 6, pp. 3275–3278, Istanbul, Turkey, June 2000.

[11] K.-H. Tan, W. F. Leong, S. Kadam, M. A. Soderstrand, and L. G. Johnson, "Public-domain matlab program to generate highly optimized VHDL for FPGA implementation," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '01)*, pp. 514–517, Sydney, Australia, May 2001.
[12] D. Brückmann, "Optimized digital signal processing for flexible receivers," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '02)*, vol. 4, pp. 3764–3767, Orlando, Fla, USA, May 2002.
[13] F. Cruz-Roldán and M. Monteagudo-Prim, "Efficient implementation of nearly perfect reconstruction FIR cosine-modulated filterbanks," *IEEE Transactions on Signal Processing*, vol. 52, no. 9, pp. 2661–2664, 2004.
[14] W. Sung and S. K. Mitra, "Implementation of digital filtering algorithms using pipelined vector processors," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1293–1303, 1987.
[15] M. D. Meyer and D. P. Agrawal, "Vectorization of the DLMS transversal adaptive filter," *IEEE Transactions on Signal Processing*, vol. 42, no. 11, pp. 3237–3240, 1994.
[16] D. Kim and G. Choe, "AMD's 3DNow!™ vectorization for signal processing applications," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '99)*, vol. 4, pp. 2127–2130, Phoenix, Ariz, USA, March 1999.
[17] J. P. Robelly, G. Cichon, H. Seidel, and G. Fettweis, "Implementation of recursive digital filters into vector SIMD DSP architectures," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '04)*, vol. 5, pp. 165–168, Montreal, Canada, May 2004.
[18] M. Van Der Horst, K. Van Berkel, J. Lukkien, and R. Mak, "Recursive filtering on a vector DSP with linear speedup," in *Proceedings of IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pp. 379–386, Samos, Greece, July 2005.
[19] A. Shahbahrami, B. H. H. Juurlink, and S. Vassiliadis, "Efficient vectorization of the FIR filter," in *Proceedings of the 16th Annual Workshop on Circuits, Systems and Signal Processing (ProRisc '05)*, pp. 432–437, Veldhoven, The Netherlands, November 2005.
[20] J. G. A. Barbedo and A. Lopes, "A new cognitive model for objective assessment of audio quality," *Journal of the Audio Engineering Society*, vol. 53, no. 1-2, pp. 22–31, 2005.
[21] J. G. A. Barbedo and A. Lopes, "A new strategy for objective estimation of the quality of audio signals," *IEEE Latin-America Transactions*, vol. 2, no. 3, 2004.
[22] ITU-R Recommendation BS-1387, "Method for Objective Measurements of Perceived Audio Quality," 1998.
[23] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, USA, 1989.

Jayme Garcia Arnal Barbedo received the B.S. degree in electrical engineering from the Federal University of Mato Grosso do Sul, Brazil, in 1998, and the M.S. and Ph.D. degrees for research on the objective assessment of speech and audio quality from the State University of Campinas, Brazil, in 2001 and 2004, respectively. From 2004 to 2005 he worked with the Source Signals Encoding Group of the Digital Television Division at the CPqD Telecom & IT Solutions, Campinas, Brazil. Since 2005 he has been with the Department of Communications of the School of Electrical and Computer Engineering of the State University of Campinas as a Researcher, conducting postdoctoral studies in the areas of content-based audio signal classification, automatic music transcription, and audio source separation. His interests also include audio and video encoding applied to digital television broadcasting and other digital signal processing areas.

Amauri Lopes received his B.S., M.S., and Ph.D. degrees in electrical engineering from the State University of Campinas, Brazil, in 1972, 1974, and 1982, respectively. He has been with the Electrical and Computer Engineering School (FEEC) at the State University of Campinas since 1973, where he has served as a Chairman in the Department of Communications, Vice Dean of the Electrical and Computer Engineering School, and currently is a Professor. His teaching and research interests include analog and digital signal processing, circuit theory, digital communications, and stochastic processes. He has published over 100 refereed papers in some of these areas and over 30 technical reports about the development of telecommunications equipment.