Audio Engineering Society

Convention Paper 5660

Presented at the 113th Convention
2002 October 5–8 Los Angeles, California, USA

Optimal Filter Partition for Efficient Convolution with Short Input/Output Delay

Guillermo García

Creative Advanced Technology Center, Scotts Valley, CA, USA.
guille@atc.creative.com
Center for Computer Research in Music and Acoustics (CCRMA), Stanford University, CA, USA.
guille@ccrma.stanford.edu

ABSTRACT

A new algorithm to find an optimal filter partition for efficient long convolution with low input/output delay is presented. For a specified input/output delay and filter length, our algorithm finds the non-uniform filter partition that minimizes computational cost of the convolution. We perform a detailed cost analysis of different block convolution schemes, and show that our optimal-partition finder algorithm allows for significant performance improvement. Furthermore, when several long convolutions are computed in parallel and their outputs are mixed down (as is the case in multiple-source 3-D audio rendering), the algorithm finds an optimal partition (common to all channels) that allows for further performance optimization.

INTRODUCTION

The direct implementation of the convolution sum in the time domain has no inherent latency, but its computational cost - measured as the number of multiply-add operations - per output sample increases linearly with the length of the convolving filter [4], which makes this algorithm impractical for performing long convolutions in real-time.

On the other hand, frequency-domain single-block convolution based on the overlap-add or overlap-save schemes [7] has a cost per output sample that increases only logarithmically with the length of the convolving filter. However, this high efficiency comes at the expense of an input/output delay equal to at least the impulse response length [4].

A common approach to achieve low latency while keeping computational cost down is to partition the convolving impulse response into shorter blocks [1]. The filter can be represented by an equivalent set of shorter filters in parallel, as represented in figure 1. Each parallel branch consists of one block of impulse response, and a delay equal to the time-offset of that block into the impulse response. Single-block convolution is performed independently for each branch, and the branch outputs are overlap-added.

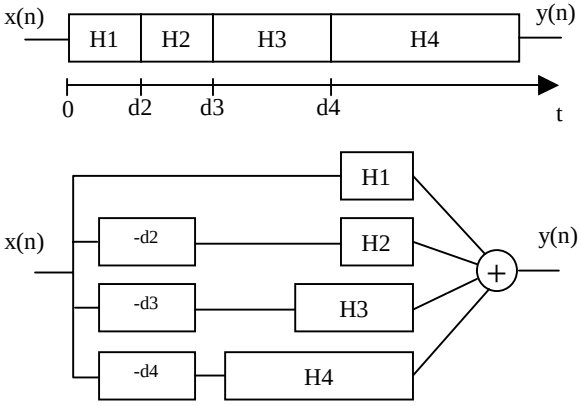

Figure 1: Long filter partitioned into four blocks (top), and equivalent parallel structure of four shorter filters (bottom).

The simplest scheme of this kind consists of a partition into blocks of uniform length. In this case, only one FFT on the input is needed for each output sample block, since single-block convolutions in each branch use the same block length. If the partition is made into blocks of different sizes, then one FFT on the input is needed for each block size.

[4] presents an efficient multiple-block algorithm based on a particular non-uniform partition into blocks of increasing size, with shorter blocks heading the impulse response. This exploits the fact that short blocks provide low latency, whereas longer blocks make the convolution less expensive. This additional degree of freedom makes this scheme generally more efficient than uniform partition.

However, uniform partition offers room for performance optimization if the overlap-add is performed in the frequency-domain. FFT blocks after spectral multiply can be overlap-added directly in the frequency domain [2][3], i.e. onto a "frequency-domain delay line" (FDL), and then only one inverse FFT needs to be performed for each input FFT. This algorithm has a convex cost function and the optimal block size can be obtained by derivation. However, for long filters the optimal block size is usually too long compared to acceptable input/output delay values, and the cost increases dramatically when block length is shortened.

One solution to this is to partition the filter into two FDLs, i.e. two sets of uniform-length blocks: a header FDL of short block size fixed by the latency requirement, followed by a second FDL of longer block size to keep cost down. In our paper we show how to optimize the cost of this algorithm by varying the block size and number of blocks of the second FDL, and the number of blocks of the header FDL. The cost of this "double-FDL convolution" algorithm remains at much lower levels for short input/output latencies.

The double-FDL approach suggests that further performance optimization could be achieved with a partition into multiple FDLs, i.e. into multiple segments where each segment consists of a set of uniform-length blocks, each segment having a block length larger than the previous segment. The parameters of this partition scheme are the total number of FDLs and the block size and number of blocks of each FDL.

However, for impulse responses several seconds long and low specified input/output latencies, the number of possible multiple-FDL partitions is quite large and it is not trivial how to choose an efficient one. Performing an exhaustive search over all possible partitions can be computationally very expensive, especially when the partition needs to be updated periodically in order to track variable filter lengths, and there is little hope that an arbitrarily picked partition be efficient.

In this paper, we present an efficient algorithm to automatically find the optimal multiple-FDL partition that minimizes computational cost of the convolution, for a given filter length and a specified input/output delay. The algorithm uses dynamic programming and allows for optimization of several convolution channels in parallel, where further performance improvement can be achieved by downmixing the channels in the frequency-domain, if the same partition is used for all channels.

Cost comparisons show that multiple-FDL convolution, based on the optimal partition found by our algorithm, is more than twice as efficient as the non-uniform block convolution algorithm given in [4].

In the following sections we present a cost analysis of the main frequency-domain block convolution schemes, and then describe how to find the optimal partition for the multiple-FDL scheme.

COST ANALYSIS OF FREQUENCY-DOMAIN BLOCK CONVOLUTION ALGORITHMS

In the following cost analysis, we assume that the Discrete Fourier Transform is computed using the FFT algorithm. The cost to compute a N-point real

FFT is assumed to be of the form $k.N.log_2(N)$, where k is a proportionality constant that depends on the particular FFT implementation available. We will measure computational cost as number of multiply-adds ("madds") per output sample, and designate it by O_{symbol} where *symbol* designates the particular algorithm measured. The FFT of the impulse response can be pre-computed and thus is not taken into account in the cost calculations.

Single-Block Convolution

For an impulse response of N points, both the overlap-add and the overlap-save schemes [7] need to calculate FFT blocks of 2N points in order to avoid time-aliasing in the output block. At each block operation, N new output samples are generated, since the rectangular window on the input signal slides N samples at a time. The cost per output sample is then obtained by normalizing the cost of a block operation by N. The cost of the direct FFT is:

$$O_{FFT} = k\,(2N)\,log_2(2N) / N = 2k\,log_2(2N)$$

The resulting FFT block of N complex points is then multiplied by the FFT block of the impulse response. This spectral multiply requires N complex multiplications, or 4N real multiply-adds:

$$O_{SpecMult} = 4N / N = 4$$

The FFT-1 on the resulting block has the same cost as the direct FFT on the input signal. In the overlap-add scheme, an additional cost of N adds (i.e. one add/sample) is necessary to overlap-add the first N-point half of the 2N-point output block, while overlap-save just saves the second N-point half of the 2N-point output block. We will use the overlap-save scheme in the rest of this document. Then, the cost of single-block (SB) convolution is:

$$O_{SB}(N) = O_{FFT}+O_{SpecMult}+O_{FFT-1} = 4k\,log_2(2N) + 4$$

For a typical value of k=1.5, we can verify that single-block convolution is less costly than direct-sum time-domain convolution for $N \geq 64$ points.

Multiple-Block Convolution with Uniform Partition

In this scheme, the filter impulse response of length T is partitioned into blocks of same length N. For simplicity, let's suppose N is an integer power of two and T/N is an integer (the impulse response can be zero-padded as necessary to satisfy this condition).

To generate an output block, only one 2N-point direct FFT is necessary since all blocks in the partition have same length. Then, a spectral multiply and a 2N-point inverse FFT are performed for each of the (T/N) blocks in the partition, and the resulting (T/N) sample blocks are overlap-added with the corresponding delays onto the output buffer. Therefore, the total cost of multiple-block with uniform partition convolution (MBUP) is:

$$O_{MBUP}(N) = O_{FFT} + (T/N)\,(O_{FFT-1} + O_{SpecMult} + O_{OvAdd})$$

$$= 2k\,log_2(2N) + (T/N)\,[2k\,log_2(2N) + 4 + 1]$$

$$= 2k\,log_2(2N)\,(1+T/N) + 5\,T/N$$

A plot of this equation (Figure 2) shows that, for a given filter length T, the cost varies monotonically with respect to the input/output latency N. Reducing N comes at the expense of increased cost, the minimum cost being achieved with no partition at all (i.e. single-block convolution).

Multiple-Block Convolution with Non-Uniform Partition

In this case two possible partition schemes are used [4]: a minimum-cost algorithm uses a partition into (I+1) blocks of size

$$N, N, 2N, 4N, 8N, \ldots, 2^{(I-1)}N$$

Another algorithm, designed to achieve uniform processor load, partitions the filter into (2I) blocks of size

$$N, N, 2N, 2N, 4N, 4N, \ldots, 2^{(I-1)}N, 2^{(I-1)}N$$

When null input/output delay is desired, [4] uses direct-sum convolution in the time-domain for the header block of length N. For fair comparison with the other partition schemes in this paper, we will use frequency-domain single-block convolution instead, which is typically cheaper for values of N equal or greater than 64 samples.

The minimum-cost algorithm can be seen as a superposition of single-block convolutions for the block series N, 2N, 4N, 8N, …, $2^{(I-1)}$N, plus the additional spectral multiply and inverse FFT corresponding to the header block of length N (that shares the input direct-FFT with the second block, also of length N). Therefore, the cost of this algorithm (denoted by symbol MBMC) is:

$$O_{MBMC}(N) = O_{SpecMult}(N) + O_{FFT-1}(N) + \sum_{i=0}^{I-1} O_{SB}(2^i N)$$

where

$$\begin{aligned}\sum_{i=0}^{I-1} O_{SB}(2^i N) &= \sum_{i=0}^{I-1} [4k \log_2(2^{i+1} N) + 4] \\ &= I(4k \log_2 N + 4) + 2kI(I+1) \\ &= I(4k \log_2 N + 2k + 4) + 2kI^2\end{aligned}$$

Since the filter length is $T = N + \sum_{i=0}^{I-1} 2^i N = N.2^I$, we have $I = \log_2(T/N)$, and the cost of this algorithm is:

$$\begin{aligned}O_{MBMC}(N) &= 4 + 2k \log_2(2N) \\ &\quad + \log_2(T/N)(4k \log_2 N + 2k + 4) \\ &\quad + 2k[\log_2(T/N)]^2\end{aligned}$$

The cost of the uniform-load algorithm (denoted MBUL) can be calculated in a similar way and is:

$$\begin{aligned}O_{MBUL} &= (6k \log_2 N + 3k + 8) \log_2\left(1 + \frac{T}{2N}\right) \\ &\quad + 3k\left[\log_2\left(1 + \frac{T}{2N}\right)\right]^2\end{aligned}$$

Multiple-Block Convolution with Uniform Partition, using a Frequency-Domain Delay Line (FDL): Single-FDL Convolution

When the block partition is uniform, i.e. all blocks have the same length, then all delays in the branches of the equivalent parallel filter structure (Figure 1) are integer multiples of the block size. Thus the blocks that are overlap-added in the time domain are aligned, that is, the output block of branch *i* at time frame *k* has a 100% overlap with the output block of branch *(i-1)* at time frame *(k+1)*. Consequently, the overlap-add procedure can be implemented directly in the frequency domain after the spectral multiply operation is performed on each branch [2], and only one inverse FFT is needed, as opposed to one inverse FFT per branch.

To generate an output block we must start by computing one 2N-point direct FFT on the input signal. Then, a spectral multiply and an overlap-add operation are performed for each of the (T/N) blocks in the partition. These two operations can be combined into 4 real multiply-adds per FFT bin. Finally, one 2N-point inverse FFT is computed on the block coming out of the FDL. The total cost of the single-FDL convolution algorithm is then:

$$O_{SingleFDL}(N) = O_{FFT} + O_{FFT-1} + (T/N)(O_{SpecMultOvAdd})$$

$$= 4k \log_2(2N) + 4(T/N)$$

As this cost equation shows, for a given filter length T the cost of the single-FDL algorithm does not necessarily vary monotonically in the interval N=[2,T], but can have a minimum for a value of N smaller than T. This happens because the FFT cost dominates for large values of N, while for small values of N the cost of the spectral multiplies becomes predominant.

We can calculate the optimal block length by relaxing the integer-power-of-two constraint on the variable N, allowing it to take any value on the positive real scale, setting the derivative of the cost with respect to N to zero, and solving for N. This yields the optimal value:

$$N_{opt} = T \,.\, \ln(2) / k$$

$$= T \,.\, 0.6931 / k$$

where ln(.) is the natural logarithm. The optimal block size can be obtained by inspection, by evaluating the cost function on the immediate inferior and superior integer-power-of-two values around N_{opt}, and picking the one for which the cost is minimum.

As an interesting remark we observe that, even when input/output latency is not an issue, partitioning the impulse response may still be desirable if the FFT implementation available is not very efficient (i.e. if k is significantly larger than ln(2)).

Finally, if parallel convolutions are to be computed on P different sound sources, and their outputs mixed down, the mix can be done in the frequency domain and only one inverse-FFT after mix down is necessary. In this case the cost per convolution channel (and per output sample) would be:

$$O_{SingleFDL}(N) = O_{FFT} + (1/P)O_{FFT-1} + (T/N)O_{SpecMultOvAdd}$$

$$= (1+1/P)\, 2k \log_2(2N) + 4(T/N)$$

Double-FDL Convolution

Although the single-FDL algorithm allows for significant performance improvement over the multiple-block algorithm with uniform partition (MBUP), its cost curve still grows dramatically when N decreases from its optimal value. When short input/output delay (and therefore small block length N) is required, the algorithm works far away from optimality and can still be too expensive. For example, if the latency specification required N=128 and the filter length were T=65536, then the cost would be of 2096 madds (assuming k=1.5), whereas the cost for N optimal (N=32768) is of 104 madds.

One solution to this is to partition the impulse response into two FDLs: a header FDL of short blocks to satisfy the latency requirement, followed by a second FDL of longer blocks to minimize the cost. The latency requirement fixes N, i.e. the block length of the header FDL. The block length of the second FDL, which we will call B, is the variable that allows us to minimize the cost for a given latency N and filter length T. For simplicity, we constrain both N and B to be integer powers of two. According to the cost formula for a single FDL, the cost of this algorithm is then:

$$O_{DoubleFDL} = 4k \log_2(2N) + 4\,B/N + 4k \log_2(2B) + 4\,(T/B - 1)$$

were the two first terms correspond to the header FDL, and the two last terms to the second FDL.
We can calculate the block length B that minimizes this cost using the same procedure of relaxation, derivation and rounding as in the previous section. The solution for the optimal value of B (before rounding) is:

$$B = -k N / [2 \ln(2)] + \mathrm{sqrt}(\{k N / [2 \ln(2)]\}^2 + T N)$$

where sqrt(.) denotes the square-root function. Figure 2 shows a cost comparison plot between multiple-block with uniform partition (MBUP) and the FDL-based algorithms.

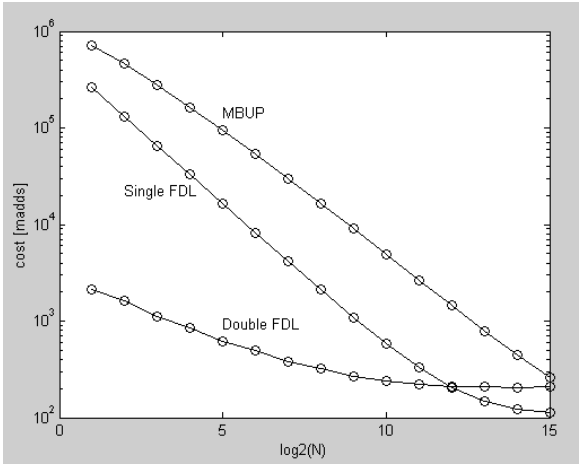

Figure 2: cost (in log scale) of multiple-block with uniform partition (MBUP) algorithm, and single-FDL and double-FDL algorithms, for different input/output delays N, a filter length of 131072 taps, and k=3/2.

Multiple-FDL Convolution

In the double-FDL partition, the first FDL's role is to satisfy the latency requirement while the second FDL minimizes the computational cost. The cost is minimal for that particular scheme, i.e. under the constraint to use only two FDLs.

Intuitively, we can see that relaxing that constraint could lead to further optimization. We can generalize the double-FDL partition into a totally flexible scheme consisting of a multiple number of FDLs [2][3].

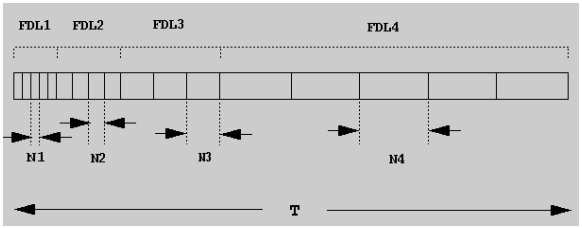

Figure 3: Non-uniform partition of an impulse response lf length T into four FDLs of block lengths N1, N2, N3 and N4 respectively.

For relatively short filter impulse responses (e.g. a few thousand of samples for adaptive filters [2]) the partition that minimizes computational cost can be found by inspection or exhaustive search. However, in applications such as reverberation or 3-D audio rendering, impulse responses several seconds long are common, and exhaustive search might not be viable. An efficient optimization procedure is therefore necessary.

For example, for a typical 3-second reverb impulse response at a 44.1HKz sample rate and a latency specification of 5.8 milliseconds (256 samples) there are 70,055,503 possible partitions. Using a typical FFT implementation (k=1.5), the cost per output sample ranges from 308 madds for the optimal partition to 2402 madds for the least-efficient partition. A trivial partition into a single FDL of 256-point blocks has a cost of 2122 madds. This suggests that an arbitrarily picked partition is not likely to be efficient, and performing an exhaustive search over all (70,055,503) possible partitions can be computationally very expensive, especially when the partition needs to be updated periodically in order to track time-varying filter lengths.

This leads us to consider more efficient ways of finding a good partition. For a given filter length T and specified input/output delay N, the multiple-FDL scheme poses an interesting cost optimization problem where the variables are:

1. The number of FDLs the filter is partitioned into.
2. The block length of each FDL except for the header FDL (whose block length is N).
3. The number of blocks of each FDL including the header FDL.

We present in the following sections a dynamics-programming algorithm we have designed to automatically find the most efficient partition for the multiple-FDL scheme, given the filter length T and the desired input/output delay N.

OPTIMAL PARTITION FINDER ALGORITHM

In order to solve the optimization problem using dynamics programming, a partition is represented as a sequence of states [5]. The state sequence that represents the optimal partition is found using the Viterbi algorithm. For a description of this well-known algorithm we refer the reader to [5][6].

The challenge in the design of the optimal-partition finder algorithm resides in defining the states and state transition costs in terms of the problem variables. Once states and transition costs are defined, the rest (i.e. the Viterbi algorithm) is a mechanical procedure that consists of evaluating partially-optimal state sequences of gradually increasing length, and finally backtracking through the optimal sequence [5][6]. In the following paragraphs we present the design of our optimal-partition finder algorithm.

States

The state sequence is defined over a set of uniformly-spaced samples along the filter impulse response of length T. This set of samples is given by n = t.N, with t integer and non-negative, and where N - an integer power of 2 - is the specified size of the header block in the partition (i.e. the input/output delay).

Let's consider a pointer moving through the set of samples n = t.N with t = 0, 1, 2, …, T/N-1; at each position t (i.e. each stage of the state sequence) we will consider all possible states of the partition. For a given position t, the state is defined by the following two parameters:

1. The size of the block the pointer falls within (e.g. a block of 4N points).
2. The fraction of the block the pointer points to (e.g. the third quarter of a 4N-point block).

We will denote the state as [S.Q] with S and Q positive integers, indicating that the pointer falls at the beginning of the Q^{th} N-point fraction of a block of size SN. For example, a state noted [4.3] indicates that the pointer falls at the beginning of the 3^{rd} quarter of a block of size 4N.
The following constraints must be respected in order to determine the possible states and possible transitions between states:

1. Block sizes must be integer powers of two.
2. A block of size S cannot start before the S^{th} sample into the filter impulse response [4].
3. A block of size S can only be followed by a block of size equal or greater than S.

Figure 4 shows all possible blocks and the corresponding state IDs at each pointer position, for a filter impulse response of length T=8N.

State-Transition Costs

For each pointer position t, we evaluate:

1. All possible states at the current position
2. All possible state transitions from states at the previous position (t-1) to the current position, and the cost of these transitions.

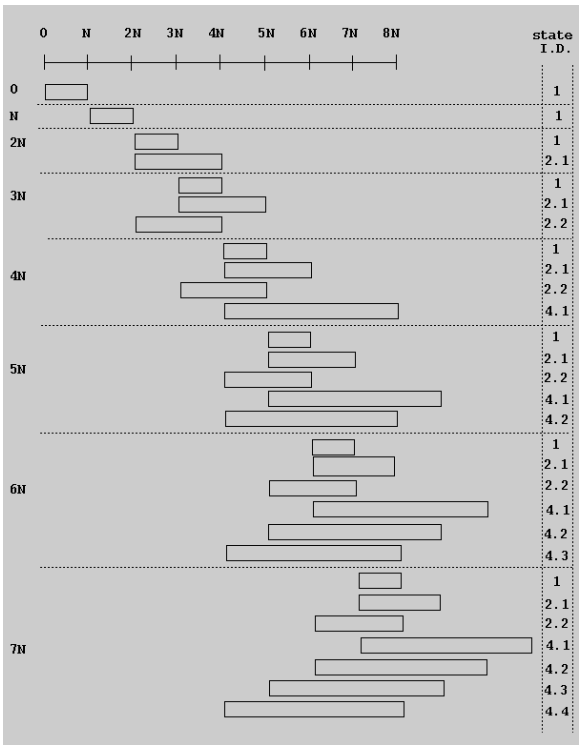

Figure 4: All possible blocks and the corresponding state IDs at each pointer position, for a filter impulse response of length T=8N.

State transition costs represent computational cost, and are defined as follows:

1. A transition into a block of greater size, i.e. from a state [X.X] into a state [Y.1] where Y>X, is a transition into a new FDL. Thus, its cost is that of one direct FFT, one spectral multiply-add and one inverse FFT:

   $cost(X.X \rightarrow Y.1;\ Y>X) = 4k \log_2(2YN) + 4$

2. If P convolutions are performed in parallel, only one inverse FFT is performed for all parallel channels, and this cost becomes:

   $cost(X.X \rightarrow Y.1;\ Y>X) = (2+2/P)\, k\log_2(2YN) + 4$

3. In the special case where Y=2X, previously-computed half-sized spectra can be used as described in [1] (roughly dividing the direct-FFT cost by a factor of two), and the transition cost becomes:

   $cost(X.X \rightarrow Y.1;\ Y>X) = (1+2/P)\, k\log_2(2YN) + 4$

4. A transition from the last fraction of a block into another block of equal size means that the current FDL is appended one more block. Thus, its cost is that of one spectral multiply-add:

   $cost(X.X \rightarrow X.1) = 4$

5. A transition between subsequent N-point fractions of the same block has no cost:

   $cost(X.Q \rightarrow X.(Q+1);\ Q<X) = 0$

Figure 5 shows the representation of states and state transitions used by our algorithm, showing which transitions are viable.

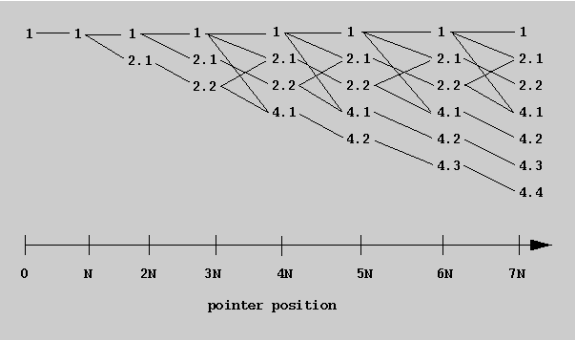

Figure 5: states and state transitions for a filter length T=8N.

Viterbi Algorithm

At each stage in the state sequence, i.e. for each position of the pointer t = 0, 1, … T/N – 1, and for each state at position t, we compute a partial accumulated cost $\delta_t(i)$ and an index $\Phi_t(i)$, where i is an index identifying the state.

For the initial position t=0 only one state is possible, i.e. state [1.1], since the partition is constrained to start by a block of length N in order to satisfy the input/output latency specification. The accumulated cost for that state, of index i=0, is:

$$\delta_0(0) = 4k \log_2(2N) + 4$$

For subsequent positions t = 1, 2, … T/N – 1, the values of $\delta_t(i)$ and $\Phi_t(i)$ are computed as:

$$\delta_t(i) = \min_j[\delta_{t-1}(j) + cost_{ji}]$$

$$\Phi_t(i) = \operatorname{argmin}_j[\delta_{t-1}(j) + cost_{ji}]$$

where index i parses all states at position t, index j parses all states at position (t-1) that can transition

into state i at position t, and $cost_{ji}$ denotes the computational cost associated with that transition, calculated as described in the previous section.

Finally, the optimal state sequence i_t (i.e. optimal state index i for each position t) is found by backtracking through the indexes $\Phi_t(i)$ starting from the state for which the accumulated cost $\delta_t(i)$ is minimum at the last position t = T/N–1:

$$i_{T/N-1} = \underset{i}{argmin}[\delta_{T/N-1}(i)]$$

and

$$i_t = \Phi_{t+1}(i_{t+1})$$

for t = T/N-2, T/N-3, …, 0.

For example, the optimal state sequence identified by the indexes i_t could look something like:

[1.1][1.1][1.1][2.1][2.2][2.1][2.2][4.1][4.2][4.3][4.4]
[4.1][4.2][4.3][4.4],

representing a partition into three FDLs, consisting of three blocks of N points, two blocks of 2N points and two blocks of 4N points respectively. In practice, of course, the algorithm deals with much longer state sequences.

RESULTS

Figure 6 shows a plot of the cost of the multiple-FDL algorithm, when the optimal partition found by our algorithm is used. For comparison, plots of the double-FDL algorithm and of the multiple-block with non-uniform partition algorithms [4] (which perform the overlap-add in the time domain) are provided.

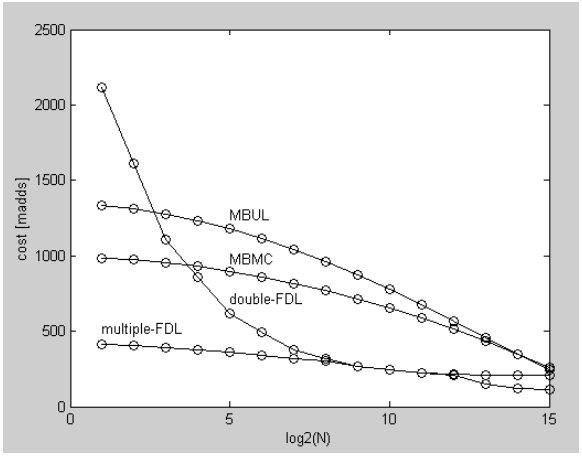

Figure 6: Cost curves for different input/output delays, showing the performance of the multiple-block with non-uniform partition algorithms [4] denoted MBUL and MBMC, the double-FDL algorithm, and the multiple-FDL algorithm using the optimal filter partition determined by our partition finder. The filter length is 131072 taps.

Figure 6 shows that the performance improvement of the multiple-FDL algorithm when using an optimal partition can be quite significant, in comparison with the other partition schemes. In particular, we see that the multiple-FDL algorithm is more than two times more efficient than the multiple-block with non-uniform partition algorithms MBMC and MBUL presented in [4].

As an illustrative example, for an impulse response 131072 samples long (almost 3 seconds at a sample rate of 44.1KHz), with a specified input/output latency of 5.8 milliseconds and a typical FFT implementation with a cost constant k=1.5, the optimal partition found by our algorithm consists of three FDLs: 8 blocks of 256 samples, 7 blocks of 2048 samples and 7 blocks of 16384 samples respectively. The computational cost per output sample of the multiple-FDL algorithm using this optimal partition is of 304 multiply-adds. If the same convolution were performed using a single block of 131072 samples (a very impractical FFT size, and a 3 sec latency) the cost would be of 112 multiply-adds. The uniform partition (MBUP) algorithm would cost 16411 madds, and the single-FDL algorithm 2102 madds per output sample. The algorithms given in [4] would cost 769 madds for the minimum-cost version (MBMC), and 964 madds for the uniform-load version (MBUL).

REFERENCES

[1] E. Ferrara, “Fast implementation of LMS adaptive filters”, IEEE Trans. on ASSP, Vol 28, No. 4, August 1980.

[2] G. Egelmeers and P. Sommen , “A new method for efficient convolution in frequency domain by nonuniform partitioning for adaptive filtering” IEEE Trans. Sig. Proc. **42**, 1994, pp. 1294-1294.

[3] D. McGrath, Lake DSP Pty Ltd., “Method and apparatus for filtering an electronic environment with improved accuracy and efficiency and short flow-through delay”, U.S. Patent #5,502,747, 1996.

[4] Gardner, W. G., “Efficient convolution without input-output delay”, J. Audio Eng. Soc. 43 (3), 127-136, March 1995.

[5] R. Bellman, “Dynamic Programming”, Princeton University Press, Boston, 1957.

[6] L. R. Rabiner, “A tutorial on hidden Markov models and selected applications in speech recognition”, Proceedings of the IEEE, 77(2):257-286, February 1989.

[7] Oppenheim and Shafer, “Digital Signal Processing”, Englewood Cliffs, NJ, Prentice-Hall, 1975.

[8] C. S. Burrus, "Efficient Fourier transform and convolution algorithms", in Advanced Topics in Signal Processing, Englewood Cliffs, New Jersey, Prentice Hall, 1988.