OPTIMAL FILTER PARTITIONS FOR REAL-TIME FIR FILTERING USING UNIFORMLY-PARTITIONED FFT-BASED CONVOLUTION IN THE FREQUENCY-DOMAIN

Frank Wefers, Michael Vorländer

Institute of Technical Acoustics,
RWTH Aachen University

{fwe,mvo}@akustik.rwth-aachen.de

ABSTRACT

This paper concerns highly-efficient real-time FIR filtering with low input-to-output latencies. For this type of application, partitioned frequency-domain convolution algorithms are established methods, combining efficiency and the necessity of low latencies. Frequency-domain convolution realizes linear FIR filtering by means of circular convolution. Therefore, the frequency transform's period must be allocated with input samples and filter coefficients, affecting the filter partitioning as can be found in many publications, is a transform size K=2B of two times the audio streaming block length B.

In this publication we review this choice based on a generalized FFT-based fast convolution algorithm with uniform filter partitioning. The correspondence between FFT sizes, filter partitions and the resulting computational costs is examined. We present an optimization technique to determine the best FFT size. The resulting costs for stream filtering and filter transformations are discussed in detail. It is shown, that for real-time FIR filtering it is always beneficial to partition filters. Our results prove evidence that K=2B is a good choice, but they also show that an optimal FFT size can achieve a significant speedup for long filters and low latencies.

Keywords: *Real-time filtering, Fast convolution, Partitioned convolution, Optimal filter partitioning*

1. INTRODUCTION

The term *fast convolution* summarizes techniques to efficiently compute the discrete convolution of two sequences x(n) and h(n). This paper considers real-time linear FIR filtering, where a *continuous stream* of input samples x(n) is convolved with an impulse response $h(n)=h_0,...,h_{N-1}$ of finite length N. Thereby a continuous stream of output samples y(n) is generated, which is defined by the discrete convolution formula

$$y(n) = \sum_{k=-\infty}^{\infty} x(k) \cdot h(n-k) \tag{1}$$

Efficient real-time FIR filtering is important for many fields in technology and science. Its use for audio applications includes real-time digital audio effects, like equalizers and convolution reverbs, audio rendering for computer games and auralization in virtual reality, comprehensive 3-D sound reproduction techniques, like wave-field synthesis (WFS)—and much more.

The filtering in eq. 1 can be easily implemented in the time-domain, using a direct-form FIR filter (transversal filter, tapped delay-line). The method has no inherent latency. Since filter coefficients can be directly altered, it allows concepts for a large variety of filter adaptation. Unfortunately, FIR filtering in the time-domain is very inefficient. The runtime-complexity for filtering N samples with N filter coefficients is within $O(N^2)$ and makes real-time filtering with long filter impulse responses virtually impossible.

Fast convolution methods

Methods that compute the discrete convolution faster than in $O(N^2)$ are known as *fast convolution algorithms*. Many of these techniques base on fast frequency-domain transforms and compute the discrete convolution within the transform's domain, where the convolution operation itself is computationally cheap to realize. The overall complexity is eventually determined by the fast frequency-domain transform algorithms, which have typical runtimes in $O(N \log N)$. The most famous transform is the Discrete Fourier Transform (DFT) implemented using Fast Fourier Transform (FFT) algorithms. But other transforms like the Discrete Trigonometric Transforms (DTTs) [1] or Number Theoretic Transforms (NTTs) [2] can be used to implement fast convolution as well, yielding to different formulations of the convolution operation in the transform's domain. A great deal of research has been spent on finding the minimum number of arithmetic operations to implement discrete convolution. For small sizes, number theoretic convolution concepts, like the Agarwal-Cooley algorithm [3], require less arithmetic operations. But for long filters FFT-based convolution is most efficient [3]. The popularity of FFT-based fast convolution algorithms is to a high degree reasoned by the availability of highly-optimized FFT libraries (e.g. FFTW [4], Intel Performance Primitives) and excellent CPU support of floating-point arithmetic, including instruction sets that ease the implementation of complex-valued operations.

When it comes to real-time filtering, simple frequency-domain convolution concepts, like the classical (unpartitioned) block convolution [5], lack efficiency. Filters need to be *partitioned* in order to combine computational efficiency and the demand for low input-to-output latencies. Splitting filters into several sub-filters of equal length is referred to as *uniformly partitioned convolution*. Highly-developed algorithms exist. We give an over-

view on the state-of-the-art for these techniques in section 4. An advantage of uniformly partitioned convolution is, that it can be easily implemented and suits the requirement of most applications. Another feature is that it allows combining frequency-domain filters—in serial or parallel [6]. However, when it comes to long filters, a non-uniform filter partitioning is more favorable [7,8,9,10]. It includes longer parts as well, which eventually lower the overall computational complexity. Unfortunately, non-uniform methods are difficult to implement and also put up additional restrictions on the exchange of filters [11]. The question for the optimal non-uniform filter partition has been raised by several authors [9,10].

2. CONTRIBUTIONS

When implementing real-time FIR filtering by partitioned convolution in the frequency-domain, one can choose the transform sizes—e.g. FFT sizes. Basically, small transform sizes K can be efficiently computed using *codelets* [4], but they have the disadvantage of resulting in many filter parts. Large size transforms on the other hand reduce the number of filter parts, but might compute less efficiently. For real-time filtering the standard choice is a transform size of K=2B two times the block length B, which can be found in numerous publications—regarding uniformly [12] and non-uniformly partitioning [9,10]. This choice seems reasonable [12], but it is unclear whether it is the optimal solution in terms of the lowest computational effort.

In this paper we therefore research the influence of the transform size used for FFT-based partitioned frequency-domain convolution and optimize it for maximum computational efficiency. The following aspects are concerned:

- The correspondence between transform sizes and the filter partitioning is derived.
- A generalized uniformly-partitioned FFT-based convolution algorithm is presented, incorporating state-of-the-art techniques. Its runtime is analyzed in detail.
- Founding on this algorithm, the FFT size is optimized for maximum computational efficiency.
- The resulting computational costs for streaming filtering and also the exchange/adaptation of filters are reviewed.
- General conclusions are drawn and the consequences to other real-time filtering techniques are discussed.

3. PROBLEM DESCRIPTION

We regard the problem of real-time FIR filtering from the perspective that filter length N and input-to-output latency, given by the audio streaming block length B, are fixed constraints. B is chosen to meet the low latency requirements–typical values are small powers of two, like 128, 256 or 512 samples. The objective is to minimize the computational effort required for the filtering. Therefore, this work concerns the FFT size K as an optimization parameter. The choice of the transform size K has consequences on the filter partitioning and the efficiency. In this section we introduce the basic relations between these variables.

In general, the convolution of two sequences with finite lengths M, N yields to a sequence with a maximum length of M+N-1. The circular convolution property of the DFT [13] states that

$$y(n) = DFT^{-1}_{(k)}\{DFT_{(k)}\{x(n)\} \cdot DFT_{(k)}\{h(n)\}\} \tag{2}$$

for a DFT size K. In order to realize the desired linear convolution in eq. 1 by circular convolution as in eq. 2, the lengths M, N of the two sequences must satisfy the critical condition

$$K \geq M + N - 1 \tag{3}$$

meaning that the maximum length M+N-1 of the result y(n) fits into the DFT period of K points. Otherwise the output is time-aliased and incorrect. For a maximal computational efficiency it is advised to fully exploit a DFT period of K values. Under-utilization of the DFT period K is unfavorable, because it introduces unnecessary zero-padding and increases the number of filter parts. As figure 1 shows, the DFT period K can be fully utilized, by allocating B values for the input samples and by using all of the remaining K-B+1 values for filter coefficients. Accordingly, the whole filter of N coefficients is split into parts of L filter coefficients determined by

$$L = K - B + 1 \tag{4}$$

The number of resulting filter parts P is given by the integer

$$P = \left\lceil \frac{N}{K - B + 1} \right\rceil \tag{5}$$

Obviously, the number of filter parts decreases with increasing DFT sizes K, because more filter coefficients can be packed into a DFT period. Valid ranges for the FFT size K are determined by

$$B < K \leq N + B - 1 \tag{6}$$

K must in any case exceed the block length K>B, otherwise no filter coefficients are processed. For K≥N+B-1 the filter remains unpartitioned. Values K>N+B-1 are useless, because they increase costs by processing ineffective, padded zeros.

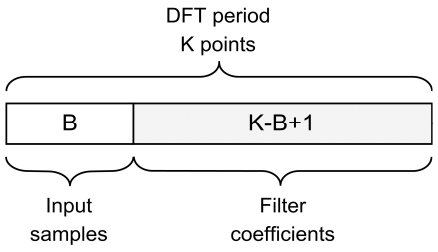


Figure 1: *Effective utilization of a full DFT period for fast circular convolution without time-aliasing.*

The simplest case is not to partition the filter and process it as a whole (P=1). Therefore, the DFT size K is chosen so that input block and filter fit into the period, meaning $K_{unpart.}$=B+N-1. The unpartitioned method is efficient for filter length N≈B close to the block length B, ideally N=B. For large N>>B far too much ineffective zeros are processed, making the method inefficient.

4. CONVOLUTION ALGORITHM

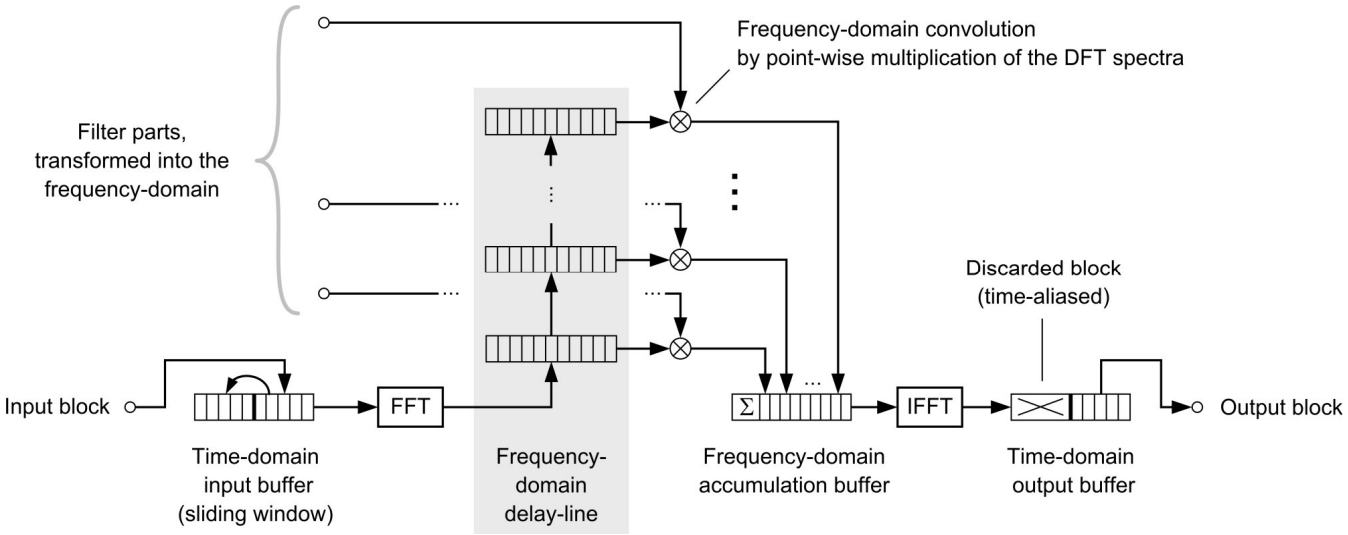
For our research of optimal FFT sizes we introduce a generalized partitioned convolution algorithm that allows FFT sizes to be freely adapted. It is illustrated in figure 2. The algorithm uses a uniform filter partitioning as discussed before. Input parameters are the block length B, corresponding to the desired input-to-output latency, and the filter length N. The DFT period of K points gets fully utilized with B input samples and L=K-B+1 filter coefficients. Accordingly, the filter of N coefficients is split into P parts of L=K-B+1 coefficients each. The algorithm incorporates several improvements, which have been published in recent years. The Overlap-Save scheme [13] is used to filter consecutive stream blocks. Overlap-Save computes more efficiently than Overlap-Add, because it saves extra additions of the partial outputs. Necessary delays for the subfilters, are directly implemented in the frequency-domain, using a frequency-domain delay-line (FDL) [14]. This is possible, because all DFT spectra share the same size. An FDL is implemented as a shift register of DFT spectra. Moreover, it is beneficial to implement the summation of the subfilters' results in the frequency-domain as well. Using these two techniques, only one FFT and one IFFT have to be computed for each processed stream block. Thereby the major computational load goes back to the complex-valued multiplications. Specialized FFTs/IFFTs for real-valued input data [15] are used and all computations are performed on complex-conjugate symmetric DFT spectra, speeding up the processing by nearly a factor of two.

The algorithm consists of two main parts: filtering the samples of the audio streams, referred to as *stream processing*. Before they can be used with the method, a *filter transformation* has to be performed, which transforms the filter impulse responses into the according partitioned frequency-domain representation. Therefore, it is uniformly partitioned into filter parts of the length L. Each filter part is zero-padded to match the FFT size K. Afterwards, each padded filter part is transformed using a K-point real-to-complex FFT.

Each block of the audio stream is processed in the following way: A time-domain input buffer acts as a sliding window of K samples on the stream of input samples. With each new input block, its contents are shifted K-B elements to the left and the new input block of B samples is then placed to the right. The whole buffer is then transformed using a K-point real-to-complex FFT and the resulting DFT spectrum is stored in a *frequency-domain delay-line* (*FDL*). Before this step, the FDL is shifted by one slot. All DFT spectra in the FDL are now point-wise complex-valued multiplied with the corresponding DFT spectra of the transformed filter parts. All results are summed up in a frequency-domain accumulation buffer. Next the contents of this buffer are transformed back into the time-domain using a K-point complex-to-real IFFT. The B left values form the output block. The other K-B values are time-aliasing and discarded.

Runtime analysis

We account the computational complexities by numbers of required arithmetic operations. These measures found on theoretical considerations. Under knowledge of the properties of the given hardware, they can be approximately translated into CPU cycles or runtimes. An exact mapping however is nearly impossible to achieve, because the runtime behaviour is hard to predict—due to cache utilization and efficiency under load of multiple threads. We assume that a K-point Fast Fourier Transform (forward and backward) can be computed with $k \cdot K \log K$ arithmetic operations (with log the natural logarithm). k is a scaling factor that depends on the actual FFT algorithm used. We benchmarked the single-threaded execution of real-valued FFTs using the FFTW3 library on an Intel Core2 system. For input sizes that are powers of two, we obtained a value of $k \approx 1.7$—assuming one arithmetic operation per CPU cycle. This scaling factor is also a good approximation for the number of arithmetic operations of the real-valued split-radix FFT in [15]. Allowing for an effective analysis, we consider idealized costs of the FFT with a constant k=1.7 for arbitrary input sizes K in the following.



A complex-valued multiplication of the form $(a+bi)(c+di) = (ac-bd)+(ad+bc)i$ requires six arithmetic operations (four multiplications and three additions). DFT spectra $X(k)$ of purely real-valued input sequences $x(n)$ fulfil the Hermitian symmetry $X(k) = \overline{X(K-k)}$ [13], for a transform size of N. This symmetry can be exploited for improved performance, because only the number of

$$\left\lceil \frac{K+1}{2} \right\rceil \tag{7}$$

symmetric DFT coefficients out of the total N DFT coefficients need to be stored and processed. The complex-valued multiplication of two symmetric DFT spectra therefore takes $6\lceil (K+1)/2 \rceil$ operations. The accumulation of DFT spectra is realized by point-wise additions of the elements, which accounts to $2\lceil (K+1)/2 \rceil$ operations. Note that for the accumulation of P spectra only (P-1) spectrum additions need to be carried out. Measures shall be independent of the block length and are therefore divided by B.

The computational cost for filtering *one output sample* is hence given by

$$T_{stream}(N,B,K) := \frac{1}{B}\left[2kK\log(K) + 6\left\lceil \frac{N}{K-B+1} \right\rceil \left\lceil \frac{K+1}{2} \right\rceil + 2\left(\left\lceil \frac{N}{K-B+1} \right\rceil - 1 \right) \left\lceil \frac{K+1}{2} \right\rceil \right] \tag{8}$$

The overall number of arithmetic operations for transforming a filter into the frequency-domain representation, demanding to compute P K-point FFT transforms, is expressed by

$$T_{ftrans}(N,B,K) := kK\log(K)\left\lceil \frac{N}{K-B+1} \right\rceil \tag{9}$$

5. OPTIMIZATION PROBLEM

For input parameters (N, B) the optimization problem is the minimization of the cost $T_{stream}(N,B,K)$ (eq. 8) for feasible transform sizes in the range $B < K \le N+B-1$ (eq. 6). Before discussing optimal solutions in general, we like to illustrate the characteristic properties of the cost function $T_{stream}(N,B,K)$ by the help of an example: The black curve in figure 3a shows the computational costs of the algorithm depending on the FFT size K for the example of N=4096 and B=128. For a better understanding, we also added the corresponding number of filter parts in the diagram (gray curve). For all problem instances (N, B) we found this common type of cost progression. Very small FFT sizes K just above the lower bound K>B+1 result in large numbers of filter parts, which are computationally inefficient. With increasing FFT sizes the cost decreases until the optimum K_{opt} is reached—in the example K_{opt}=443. From here on the costs increase again. Generally, the number of filter parts given by eq. 5 decreases with increasing FFT sizes K until it reaches the minimum of 1 for K≥N+B-1 (indicated by the vertical line). From this point on the filter is unpartitioned. Larger values of K are meaningless, because unnecessary zeros are processed and the number of DFT coefficients increases. The result is a strictly linear cost progression for K≥N+B-1. From the graph we can already see, that a partitioned convolution outperforms an unpartitioned filtering clearly.

Figure 3b shows the region around the optimum K_{opt}. In between the points of discontinuity, the progression is monotonously increasing and it shows ripples. These originate from ceiling in the definition of the number of symmetric DFT coefficients in eq. 7. Local minima of the cost function are located at values of K, where in eq. 7 K+B-1 is a factor of N—or in other words, the filter size N is a multiple of the filter part length L. At these points denoted by $K_{min}^{(i)}$ the number of filter parts is reduced by one, compared to the preceding transform size $K_{min}^{(i)}$-1. This abruptly reduces the number of necessary complex-valued

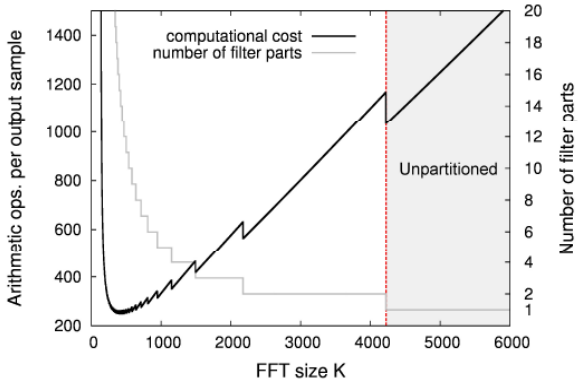


Figure 3a: *Computational costs for a fixed filter length N=4096 and block length B=128 as a function of the FFT size K (black curve) The gray curve is the corresponding number of filter parts. For K≥4332 the filter consists of a single part only and remains unpartitioned.*

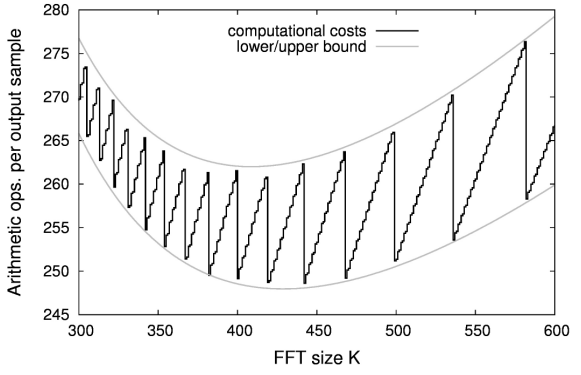


Figure 3b: *Detailed view of the cost function shown in (3a) around the optimum (cost minimum), here K_{opt}=443.*

Filter	Block	Optimal DFT size			Standard DFT size				Unpartitioned filter		
length	length	K_{opt}	parts	cost	K=2B	parts	cost	ratio	$K_{unpart.}$	cost	ratio
1024	128	298	6	99,0	256	8	100,2	1,01	1151	242,5	2,45
1024	256	460	5	71,7	512	4	72,5	1,01	1279	136,5	1,90
1024	512	767	4	56,3	1024	2	61,2	1,09	1535	83,8	1,49
1024	1024	1365	3	47,4	2048	1	57,9	1,22	2047	57,8	1,22
4096	128	443	13	248,6	256	32	293,7	1,18	4223	1035,5	4,16
4096	256	665	10	158,9	512	16	168,9	1,06	4351	535,1	3,37
4096	512	1097	7	108,9	1024	8	109,3	1,00	4607	285,1	2,62
4096	1024	1843	5	80,2	2048	4	81,9	1,02	5119	160,2	2,00
4096	2048	3071	4	63,4	4096	2	70,6	1,11	6143	98,0	1,54
4096	4096	5461	3	53,7	8192	1	67,3	1,25	8191	67,3	1,25
16384	128	713	28	743,6	256	128	1067,7	1,44	16511	4646,3	6,25
16384	256	1075	20	431,7	512	64	554,4	1,28	16639	2342,9	5,43
16384	512	1604	15	263,7	1024	32	301,6	1,14	16895	1191,2	4,52
16384	1024	2513	11	170,9	2048	16	178,0	1,04	17407	615,4	3,60
16384	2048	4095	8	118,5	4096	8	118,6	1,00	18431	327,5	2,76
16384	4096	6826	6	88,4	8192	4	91,3	1,03	20479	183,8	2,08
16384	8192	12287	4	70,5	16384	2	80,0	1,13	24575	112,1	1,59
16384	16384	21845	3	60,0	32768	1	76,7	1,28	32767	76,7	1,28
65536	128	1257	58	2508,6	256	509	4139,5	1,65	65663	20885,9	8,33
65536	256	1745	44	1366,6	512	256	2096,4	1,53	65791	10465,0	7,66
65536	512	2559	32	768,4	1024	128	1071,1	1,39	66047	5254,6	6,84
65536	1024	4002	22	450,4	2048	64	562,3	1,25	66559	2649,4	5,88
65536	2048	6143	16	278,0	4096	32	310,7	1,12	67583	1346,8	4,85
65536	4096	9557	12	182,4	8192	16	187,3	1,03	69631	695,5	3,81
65536	8192	16383	8	128,0	16384	8	128,0	1,00	73727	370,0	2,89
65536	16384	27306	6	96,2	32768	4	100,7	1,05	81919	207,3	2,15
65536	32768	49151	4	77,6	65536	2	89,4	1,15	98303	126,3	1,63
65536	65536	87381	3	66,2	131072	1	86,1	1,30	131071	86,1	1,30

Table 1: *Resulting stream filtering costs of o*ptimal FFT sizes in comparison the other methods.

multiplications. Optimal FFT sizes K_{opt} can hence easily be found by just inspecting the absolute minimum at all points $K_{min}^{(i)}$ in the interval [B+1, N+B-1].

However, it is desirable to obtain a closed formula for $K_{opt}(N, B)$ as a function of the problem instance (N, B). This is much hindered by the discontinuous ceil functions in the cost formulation (eq. 8), which demand a piecewise analysis. The problem can be relaxed by replacing ceil(x) in eq. 8 with its lower and upper bounds $x \le \lceil x \rceil \le x+1$ (see figure 3b). This yields to a continuous cost formulation of the functional form

$$f(x) = ax\log x + \frac{b}{x-c}$$

The absolute minimum of this continuous function is located at the zero of its derivative

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} = a\log x + a - \frac{b}{(x-c)^2}$$

Finding the root of this type of function turned out to be difficult as well and there does not seem to be an analytic expression for x solving $\mathrm{d}f(x)/\mathrm{d}x = 0$ (within intervals of interest), which eventually define the optimal FFT size K_{opt}.

6. RESULTS

We reviewed a multitude of problem instances (N, B) and inspected the resulting optimal transform sizes K_{opt}. In the following the results are discussed with respect to several aspects including the sheer computational cost for filtering the audio stream but also the complexity for transforming filters into the frequency-domain representation in order to use them.

Costs of stream filtering

Table 1 gives a detailed insight into the results. The two leftmost columns define the problem instance (N, B). Followed by the data of the optimal uniformly partitioned convolution using the presented algorithm, starting with the optimal FFT size K_{opt}, the

number of resulting filter parts P and the computational costs. All cost measures in the table refer to the definition in the previous section. The next block of columns lists data for uniformly partitioned convolution with the standard FFT size K=2B, two times the block length B. The fourth column in this block is the cost ratio of this method in relation to the optimal approach, given by $T_{stream}(N,B,2B)/T_{stream}(N,B,K_{opt})$. The right block lists data for the unpartitioned convolution of the instance (N, B). Here the FFT size is chosen $K_{unpart.}=N+B-1$ and the filter consists of a single part only. The rightmost column is the cost ratio of an unpartitioned filtering in comparison to the optimal solution.

Straightaway we see that for all problem instances the two cost ratios are above one. A closer comparison on the computational costs reveal, that the optimized method is faster in any case—sometimes just by a tiny margin. The data in table 1 underlines, that an unpartitioned convolution is only efficient when N≈B. Here the choice of K=2B converges against the FFT size $K_{unpart.}=N+B-1\approx 2B-1$, resulting in almost identical computational costs. We see that optimal FFT sizes K_{opt} can be smaller or larger than the FFT sizes for the two other methods. Interestingly, even for large N≈B the computational costs for an optimal FFT size K_{opt} are significantly less than for an unpartitioned filtering. Optimal filter partitions consist here of three parts and we can identify a speedup of ≈1.3 for the optimized method over the other methods. This is a very important discovery, stating that it is always beneficial to partition filters for real-time filtering in the frequency-domain.

Another observation is that for a filter length N, we can always find a block length B, where an FFT size of K=2B results in almost optimal (minimal) costs, even if K=2B differs from K_{opt}. These cases (N, B) seem to approximately fulfil N≈16B. For problem instances around this point, the standard solution K=2B drops in efficiency. But we like to point out, that the penalty in costs is rather low, proving evidence that K=2B is a very good choice in general. Nevertheless, when filtering very long filters with low latencies, an optimized FFT size K_{opt} leads to a significant reduction of costs. An example is the case of N=65536, B=128 where the speedup against the standard solution reaches 65%.

Costs of filter transformation

In case that filters are adapted or exchanged over time, they have first to be transformed into the corresponding frequency-domain representation. This computation introduces a separate latency, we refer to as *filter exchange latency*. It as well depends on the partitioning, as eq. 9 shows.

In figure 4 we compare filter transformation costs for an unpartitioned filter with the standard choice of K=2B and the optimal transform size K_{opt}. In this example a block length of B=128 was chosen. The filter transformation for a transform size K_{opt} is significantly cheaper than for K=2B. For N=4096 we find a decrease in costs ≈22% and for N=65536 ≈28%. For long filters the transformation is even cheaper than for the unpartitioned case. However, there is a lower limit in filter length where the unpartitioned filter are cheaper to realize. In the example this limit is N≈4096 taps, where N≈32B.

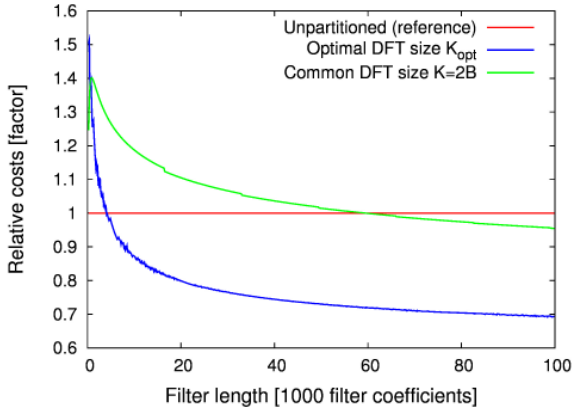


Figure 4: *Relative computational costs of the filter transformation. The graph shows factors in relation to an unpartitioned fast convolution. The block length here is B=128.*

We conclude that an optimal transform size K_{opt} lowers the computational effort for transforming filters significantly over all other methods. This is a huge advantage of our method. Not only is the stream filtering more efficient, but also the filter transformation for the majority of cases. Exceptions are found for small filter lengths. In the example we identified a maximum increase in costs of 52% for N<4096. Concerning that the transformation of small filters can be computed very fast anyway, this is no real disadvantage.

7. CONCLUSIONS

In this work we discussed the choice of transform sizes for efficient real-time linear filtering realized by fast uniformly partitioned convolution in the frequency-domain. Small transform sizes result in a large number of filter parts and vice versa. For detailed research of the optimal transform size, we presented a generalized convolution algorithm with a uniform filter partitioning and analyzed its properties. Even if they found on FFT-based implementations, our results can be applied for other transform-based convolution techniques like (e.g. Discrete Trigonometric Transforms and Number Theoretic Transforms).

The presented results give a detailed insight into the properties of optimal filter partitions for uniformly partitioned frequency-domain convolution algorithms. We can confirm that the standard transform size K=2B of twice the block length B—commonly found in literature—is generally a good choice, delivering a high computational efficiency. However, it turns out that specific optimization of the FFT size can significantly lower the costs for the case of long filters and low latencies. A very interesting observation is that the partitioning of filters is always beneficial and outperforms unpartitioned convolution in any case. We find for an optimal uniform partitioning, the computational costs do not have a linear dependency on the filter length—as it is known for non-uniformly partitioned convolution [9].

Consequences for other methods

The standard method for real-time filtering with long filters of >10.000 coefficients is non-uniformly partitioned convolution. Increasing subfilter sizes can significantly lower the computational effort compared to a uniform-partitioning. But any non-uniform filter partitioning is assembled from segments, which are basically uniformly partitioned sections with equal subfilter sizes. Consequently, our results can be applied to further optimize this class of algorithms as well and leads to an improved performance.

Applicability in practice

Optimal solutions in theory do often not translate into the desired optimal behaviour of practical implementations. A doubtful issue concerning this work might be to concern FFT sizes as an optimization parameter, without applying restrictions—for instance powers of two. A large number of FFT algorithms are known today. Efficient $O(N \log N)$ algorithms exist for arbitrary sizes (prime-factor algorithm (PFA), see [16,17]). However, FFT algorithms are most efficient if the transform size N is a highly composite number. And yet still transform sizes that are powers of two are among the most efficient. But there is no rule stating that an FFT of next greater power of two does compute faster. Therefore, it is reasonable to also account non-powers of two for implementations. Hence, our results have great importance also for practical implementations. We like to point out that the number of arithmetic operations of an FFT cannot be precisely described with a fixed scaling factor k for arbitrary input sizes. Optimal results in practice can only be achieved by benchmarking the actual runtimes on the target hardware and using these measures for the optimization.

8. REFERENCES

[1] S. A. Martucci, “Symmetric Convolution and the Discrete Sine and Cosine Transforms, “ IEEE Transactions on Signal Processing, Vol. 42, No. 5, May 1994

[2] J. Pollard, “The Fast Fourier Transform in a Finite Field,“ Mathematics of Computation, Vol. 25, No. 114, 1971, pp. 365-374

[3] R.C. Agarwal, J. W. Cooley, “New Algorithms for Digital Convolution,“ IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. 25, No. 5, May 1977

[4] Frigo, M. and S. G. Johnson, "The Design and Implementation of FFTW3," Proceedings of the IEEE 93 (2), 216–231 (2005).

[5] T.G. Stockham Jr, “High-speed convolution and correlation,” in Proceedings of the April 26-28, 1966, Spring joint computer conference. ACM, 1966.

[6] M. Joho, G.S. Moschytz, Connecting Partitioned Frequency-Domain Filters in Parallel or in Cascade, IEEE Transactions on Circuits and Systems, Vol. 47, No. 8, 2000

[7] Gerald P. M. Egelmeers and Piet C. W. Sommen, “A new method for efficient convolution in frequency domain by nonuniform partitioning for adaptive filtering,” IEEE Transactions on Signal Processing, vol. vol 44, 1996.

[8] J. S. Soo, K. K. Pang, “Multidelay Block Frequency Domain Adaptive Filter, “, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. 38, No. 2, February 1990

[9] William G. Gardner, “Efficient convolution without input-toutput delay,” Journal of the Audio Engineering Society, vol. vol 43, pp. 127–136, 1995.

[10] Guillermo García, “Optimal filter partition for efficient convolution with short input/output delay,” in Audio Engineering Society, Convention Paper 5660, 2002.

[11] Christian Müller-Tomfelde, “Time-varying filter in nonuniform block convolution,” in Conference on Digital Audio Effects (DAFX-01) proceedings, 2001.

[12] E. Armelloni, C. Giottoli, and A. Farina, “Implementation of real-time partitioned convolution on a DSP board,” 2003 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, pp. 71–74, 2003.

[13] Alan V. Oppenheim and Ronald W. Schafer, "Discrete-Time Signal Processing," Prentice Hall Signal Processing Series. Prentice Hall, 1989.

[14] Barry D. Kulp, “Digital equalization using fourier transform techniques,” Journal of the Audio Engineering Society, 1988.

[15] H. V. Sorensen, D. L. Jones, M. T. Heidman, and C. S. Burrus, “Real-valued fast fourier transform algorithms,“ IEEE Transactions on Acoustics, Speech, and Signal Processing, 1987, pp. 35:849–863,

[16] I. J. Good, "The interaction algorithm and practical Fourier analysis," J. R. Statist. Soc. B 20 (2), 361-372 (1958). Addendum, ibid. 22 (2), 373-375 (1960).

[17] P. Duhamel and M. Vetterli, "Fast Fourier transforms: A tutorial review and a state of the art," Signal Processing 19, 259-299 (1990).