

Fast Convolution Filtering -- From Basics to Filter Banks

Mark Borgerding
Xetron

Overview

- ⇒ Fast Convolution is
 - Multiplication in the frequency domain
 - A faster FIR filter for more than 10-30 taps
- ⇒ Additional channels are even cheaper
- ⇒ Other operations can be done in frequency domain
 - Downsampling/Decimation
 - Mixing*
- ⇒ Put it all together and shake it all about.
That's what Filter Banks are all about.

Goals

- ➔ Explain and demonstrate fast convolution filtering (FCF)
- ➔ Efficiently extend FCF to filter banks
 - Parallel filters
 - Mixing
 - Decimation

FIR Filtering is Convolution

$$y_n = h * x = \sum_m h_{n-m} x_m$$

- ⇒ FIR filtering arguably the most important concept in DSP
 - Stable
 - Linear phase
- ⇒ FIR filtering is another name for convolution
- ⇒ Direct FIR filter costs don't scale well
 - Need twice the work for half the bandwidth
- ⇒ Convolution of P elements with L elements is P+L-1 elements long

Visualizing Convolution

as the sum of outer product antidiagonals

	x_0	x_1	x_2	x_3	x_4
h_0	h_0x_0	h_0x_1	h_0x_2	h_0x_3	h_0x_4
h_1	h_1x_0	h_1x_1	h_1x_2	h_1x_3	h_1x_4
h_2	h_2x_0	h_2x_1	h_2x_2	h_2x_3	h_2x_4
h_3	h_3x_0	h_3x_1	h_3x_2	h_3x_3	h_3x_4

$y_0 =$	h_0x_0	Input-on Transient
$y_1 =$	$h_1x_0 + h_0x_1$	Input-on Transient
$y_2 =$	$h_2x_0 + h_1x_1 + h_0x_2$	Input-on Transient
$y_3 =$	$h_3x_0 + h_2x_1 + h_1x_2 + h_0x_3$	Steady State
$y_4 =$	$h_3x_1 + h_2x_2 + h_1x_3 + h_0x_4$	Steady State
$y_5 =$	$h_3x_2 + h_2x_3 + h_1x_4$	Input-off Transient
$y_6 =$	$h_3x_3 + h_2x_4$	Input-off Transient
$y_7 =$	h_3x_4	Input-off Transient

Circular Convolution

- ⇒ Accomplished by multiplying DFTs
- ⇒ Periodic, just like DFT

	x_0	x_1	x_2	x_3	x_4
h_0	h_0x_0	h_0x_1	h_0x_2	h_0x_3	h_0x_4
h_1	h_1x_0	h_1x_1	h_1x_2	h_1x_3	h_1x_4
h_2	h_2x_0	h_2x_1	h_2x_2	h_2x_3	h_2x_4
h_3	h_3x_0	h_3x_1	h_3x_2	h_3x_3	h_3x_4

$$\begin{aligned}
 y_0 &= h_3x_2 + h_2x_3 + h_1x_4 + h_0x_0 && \text{Wraparound Transient} \\
 y_1 &= h_3x_3 + h_2x_4 + h_1x_0 + h_0x_1 && \text{Wraparound Transient} \\
 y_2 &= h_3x_4 + h_2x_0 + h_1x_1 + h_0x_2 && \text{Wraparound Transient} \\
 y_3 &= h_3x_0 + h_2x_1 + h_1x_2 + h_0x_3 && \text{Steady State} \\
 y_4 &= h_3x_1 + h_2x_2 + h_1x_3 + h_0x_4 && \text{Steady State}
 \end{aligned}$$

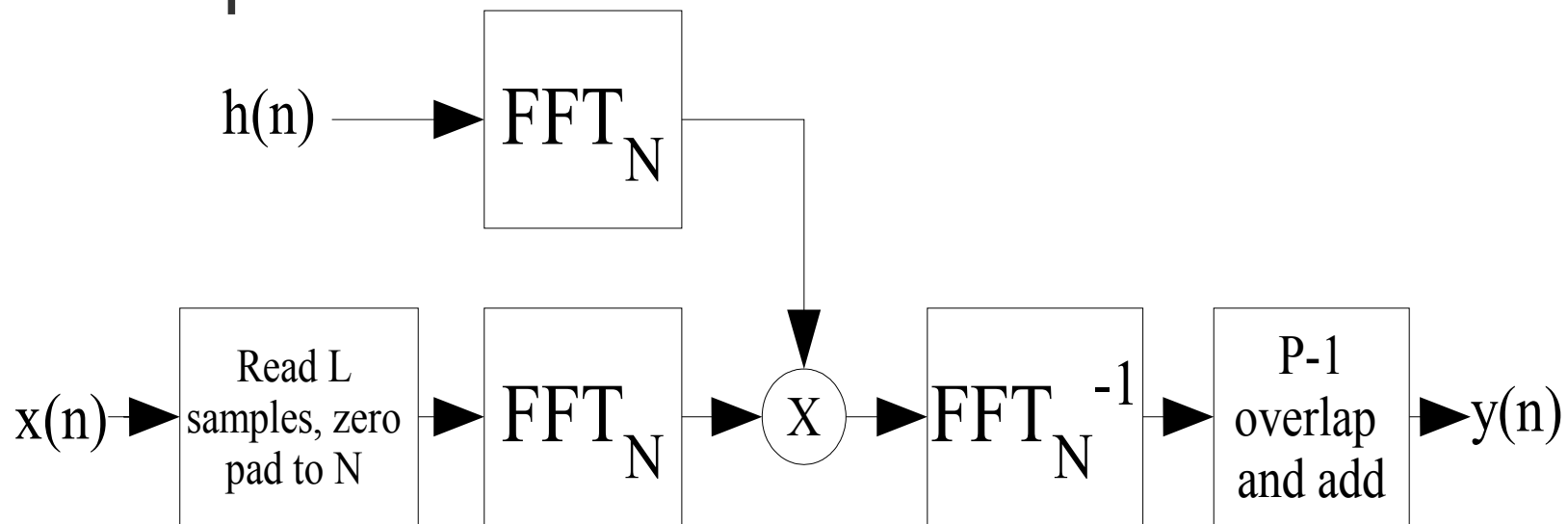
Making Circular Convolution Equivalent to Linear Convolution

- ➔ Zero-padding to $P+L-1$ avoids wraparound transient

	x_0	x_1	x_2	x_3	x_4	0	0	0
h_0	h_0x_0	h_0x_1	h_0x_2	h_0x_3	h_0x_4	0	0	0
h_1	h_1x_0	h_1x_1	h_1x_2	h_1x_3	h_1x_4	0	0	0
h_2	h_2x_0	h_2x_1	h_2x_2	h_2x_3	h_2x_4	0	0	0
h_3	h_3x_0	h_3x_1	h_3x_2	h_3x_3	h_3x_4	0	0	0

Overlap-Add Filtering

- ⇒ Builds an arbitrarily long convolution from shorter convolutions, buffer-by-buffer
- ⇒ Input-on transients from one buffer overlap (and add) with the input-off transients from the previous



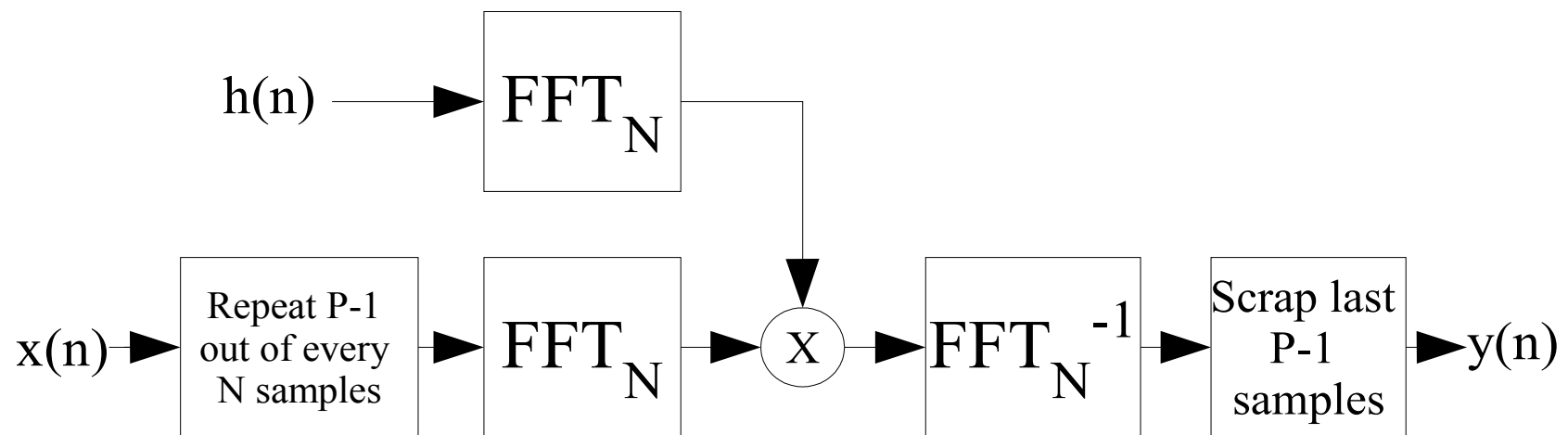
Overlap-Add (OA) Filtering

- ➔ Input-off transients complement the next buffer's input-on transients

$$\begin{array}{rcl}
 y_0 & = & \mathbf{h_0x_0} \\
 y_1 & = & \mathbf{h_1x_0} + \mathbf{h_0x_1} \\
 y_2 & = & \mathbf{h_2x_0} + \mathbf{h_1x_1} + \mathbf{h_0x_2} \\
 y_3 & = & \mathbf{h_3x_0} + \mathbf{h_2x_1} + \mathbf{h_1x_2} + \mathbf{h_0x_3} \\
 y_4 & = & \mathbf{h_3x_1} + \mathbf{h_2x_2} + \mathbf{h_1x_3} + \mathbf{h_0x_4} \\
 y'_5 & = & \mathbf{h_3x_2} + \mathbf{h_2x_3} + \mathbf{h_1x_4} \\
 y'_6 & = & \mathbf{h_3x_3} + \mathbf{h_2x_4} \\
 y'_7 & = & \mathbf{h_3x_4} \\
 y''_5 & = & \mathbf{h_0x_5} \\
 y''_6 & = & \mathbf{h_1x_5} + \mathbf{h_0x_6} \\
 y''_7 & = & \mathbf{h_2x_5} + \mathbf{h_1x_6} + \mathbf{h_0x_7} \\
 y_8 & = & \mathbf{h_3x_5} + \mathbf{h_2x_6} + \mathbf{h_1x_7} + \mathbf{h_0x_8} \\
 y_9 & = & \mathbf{h_3x_6} + \mathbf{h_2x_7} + \mathbf{h_1x_8} + \mathbf{h_0x_9} \\
 y_{10} & = & \mathbf{h_3x_7} + \mathbf{h_2x_8} + \mathbf{h_1x_9} \\
 y_{11} & = & \mathbf{h_3x_8} + \mathbf{h_2x_9} \\
 y_{12} & = & \mathbf{h_3x_9}
 \end{array}$$

Overlap-Save Filtering

- ➔ Eliminates addition of overlapping transients
- ➔ Circular convolution w/o zero padding
- ➔ Overlaps input buffers



Overlap-Save (OS) Filtering

- ➔ A.K.A. “Overlap-Scrap”
- ➔ Input buffers overlap
- ➔ wraparound transient is useless and discarded (i.e. scrapped)

$$y'_{s0} = h_3x_2 + h_2x_3 + h_1x_4 + h_0x_0 \quad \text{Wraparound Transient}$$

$$y'_{s1} = h_3x_3 + h_2x_4 + h_1x_0 + h_0x_1 \quad \text{Wraparound Transient}$$

$$y'_{s2} = h_3x_4 + h_2x_0 + h_1x_1 + h_0x_2 \quad \text{Wraparound Transient}$$

$$y_3 = h_3x_0 + h_2x_1 + h_1x_2 + h_0x_3 \quad \text{Steady State}$$

$$y_4 = h_3x_1 + h_2x_2 + h_1x_3 + h_0x_4 \quad \text{Steady State}$$

$$y'_{s2} = h_3x_4 + h_2x_5 + h_1x_6 + h_0x_2 \quad \text{Wraparound Transient}$$

$$y'_{s3} = h_3x_5 + h_2x_6 + h_1x_2 + h_0x_3 \quad \text{Wraparound Transient}$$

$$y'_{s4} = h_3x_6 + h_2x_2 + h_1x_3 + h_0x_4 \quad \text{Wraparound Transient}$$

$$y_5 = h_3x_2 + h_2x_3 + h_1x_4 + h_0x_5 \quad \text{Steady State}$$

$$y_6 = h_3x_3 + h_2x_4 + h_1x_5 + h_0x_6 \quad \text{Steady State}$$

OA & OS Comparison

	<i>Overlap-Add Filtering</i>	<i>Overlap-Save Filtering</i>
<i>FFT input buffer</i>	L input samples, zero-padded to $N=L+P-1$	N input samples, P-1 of which are repeated from the previous buffer
<i>IFFT output buffer</i>	The last P-1 samples from the previous output are added to the first P-1 samples of the current output. The first L ($L = N-P+1$) samples are then suitable for output.	The first P-1 samples of each IFFT output are discarded. The remaining N-P+1 samples are used as output.
<i>Speed</i>	Cost is logarithmic wrt. FIR length. Faster than direct for filters longer than 25-30 taps	Same scalability as OA, but slightly more efficient.
<i>Name notes</i>	The input-on and input-off transients are overlapped and added .	The input buffers are overlapping , i.e. some of the input is saved between iterations. ¹

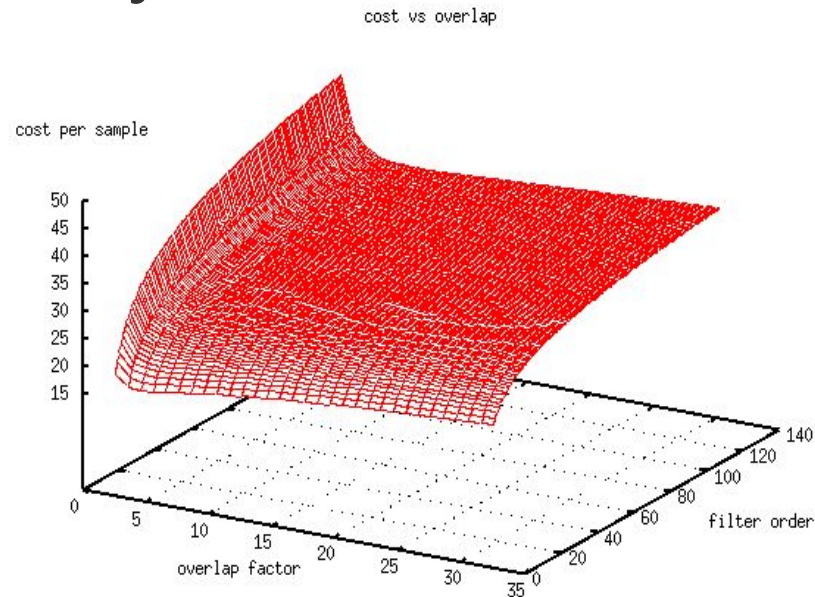
¹ Alternately, the wraparound transient is “scrapped”.

Fast Convolution Complexity

- ⇒ filter order $P'=P-1$
- ⇒ Define overlap factor $V=N/P'$
- ⇒ Cost per FFT buffer = $O(VP' \log(VP'))$
- ⇒ Throughput samples per buffer = $(V-1)P'$
- ⇒ Cost per sample = $O(V \log(VP') / (V-1))$
- ⇒ For a given overlap factor V
 - V determines multiplier against big “O”
 - Cost per sample = $O(\log(P'))$

Determining Overlap Factor V

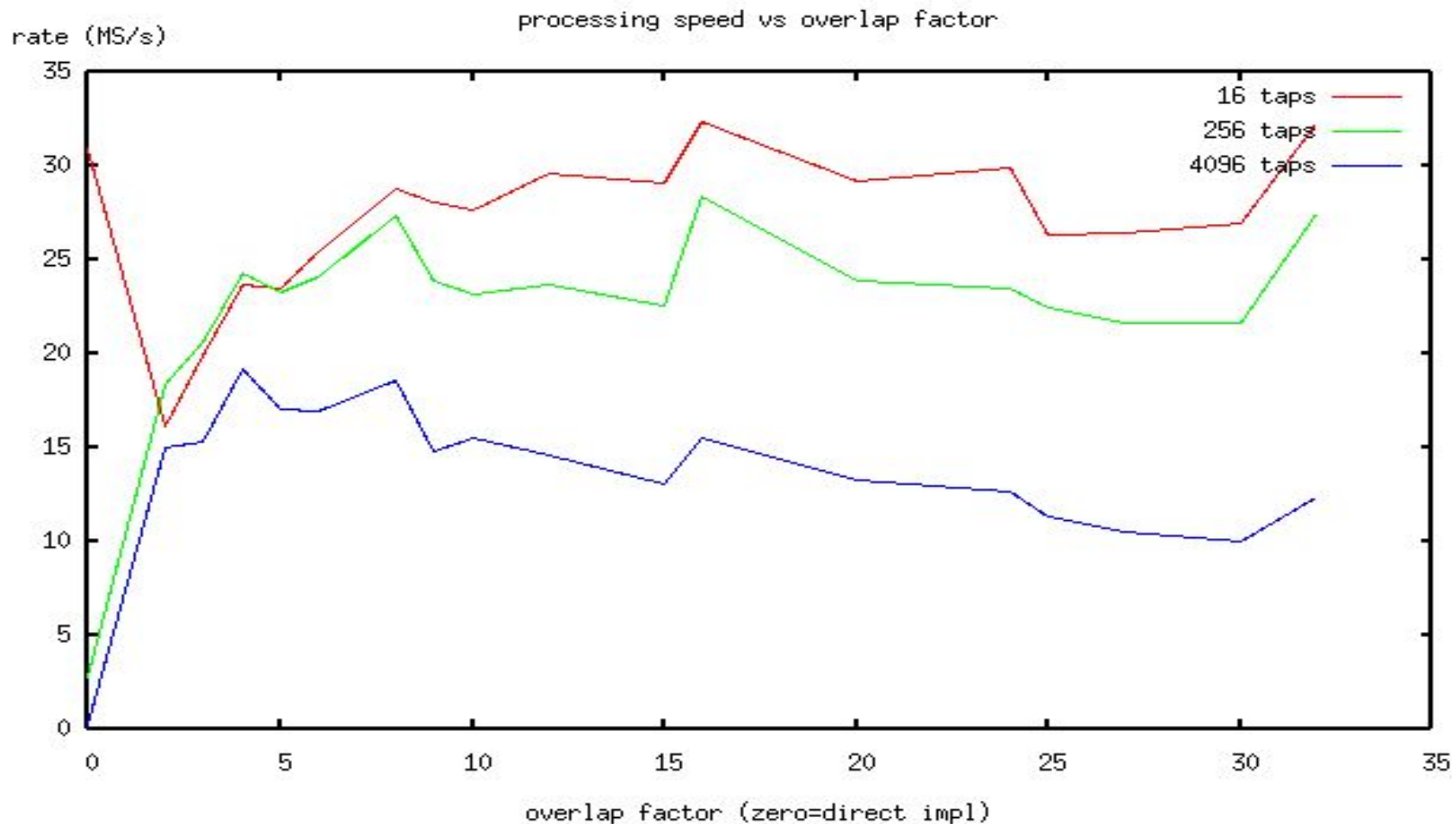
- ⇒ Cost per sample is $O(V \log(VP') / (V-1))$
- ⇒ For a given filter order P' , how is the fastest V determined?
- Theoretically: find minimum of $V \log(VP') / (V-1)$



- Empirically: Benchmark!

Overlap Factor Efficiency

- ➔ Theory vs. practice
- ➔ If possible, compare FCF and direct FIR filtering (shown at 0 in figure)



Recap

- ➔ Multiplication in the frequency domain is equivalent to convolution in time domain
- ➔ Circular Convolution is faster (i.e. more scalable) than linear convolution
- ➔ Fast Convolution combines smaller convolutions to calculate a longer convolution