Fast Time-Domain Volterra Filtering

HARALD ENZINGER[1] KARL FREIBERGER[1] GERNOT KUBIN[1] CHRISTIAN VOGEL[1,2]
enzinger@tugraz.at freiberger@tugraz.at g.kubin@ieee.org c.vogel@ieee.org
[1]Signal Processing and Speech Communication Laboratory, Graz University of Technology, Austria
[2]FH Joanneum - University of Applied Sciences, Austria

Abstract

- **Nonlinear filtering with less multiplications**
- **Fast method for input products (1 mult./product)**
- **Fast method for output samples (1 mult./param.)**

1. Introduction

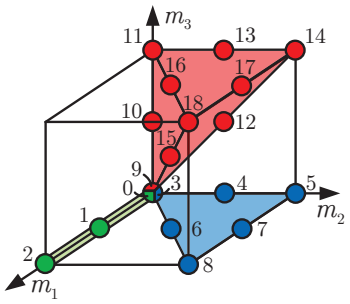
- Volterra series combines convolution and polynomial

$$y[n] = \sum_{p=1}^{P} \sum_{m_1=0}^{M} \cdots \sum_{m_p=0}^{M} h_p[m_1, \ldots, m_p] \prod_{l=1}^{p} x[n - m_l]$$

- Universal approximator for nonlinear systems with memory

2. Methods of Traversing

- Iterate through non-redundant input products
  - Nested loop
  - Combinatoric
  - Lookup-table
  - Hard-coded



i . . . kernel index
j . . . reuse / projection index

p	i	j	m_1	m_2	m_3
1	0	–	0	–	–
	1	–	1	–	–
	2	–	2	–	–
2	3	0	0	0	–
	4	0	0	1	–
	5	0	0	2	–
	6	1	1	1	–
	7	1	1	2	–
	8	2	2	2	–
3	9	3	0	0	0
	10	3	0	0	1
	11	3	0	0	2
	12	4	0	1	1
	13	4	0	1	2
	14	5	0	2	2
	15	6	1	1	1
	16	6	1	1	2
	17	7	1	2	2
	18	8	2	2	2

3. Methods of Computation

- Reuse - computes input products & output samples

$$y[n] = \sum_{i=0}^{i_{\max}} h[i]\, \Phi_i[n] \qquad \Phi_i[n] = \Phi_j[n]\, x[n - m_p]$$

- Horner - computes output samples only

$$y[n] = \sum_{m_1=0}^{M} x[n - m_1]\, g_1[n, m_1]$$

$$g_1[n, m_1] = h_1[m_1] + \sum_{m_2=m_1}^{M} x[n - m_2]\, g_2[n, m_1, m_2]$$

$$g_2[n, m_1, m_2] = h_2[m_1, m_2] + \sum_{m_3=m_2}^{M} x[n - m_3]\, g_3[n, m_1, m_2, m_3]$$

$$\vdots$$

$$g_P[n, m_1, \ldots, m_P] = h_P[m_1, \ldots, m_P]$$

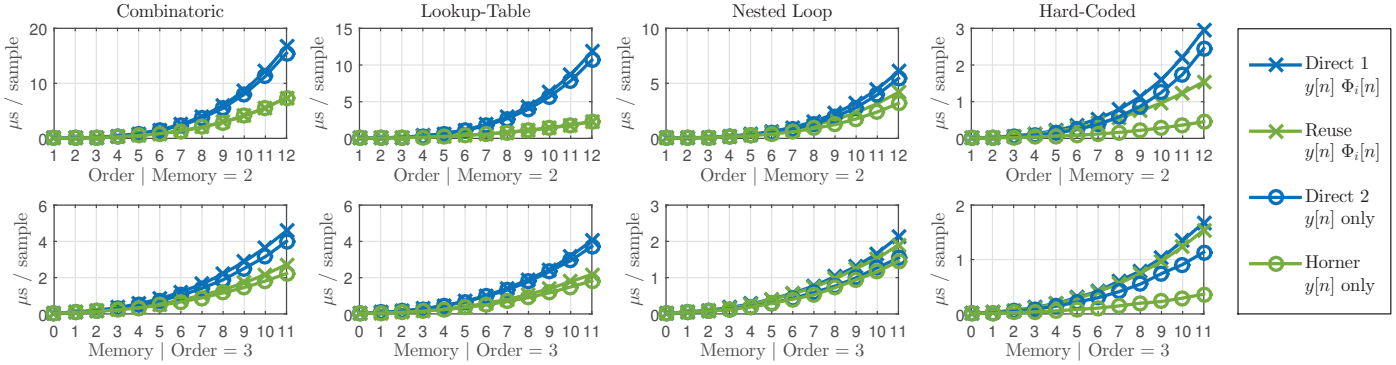
p	j	m	i	Operation	Formula
3	3	0	9	$g[3] \leftarrow h[3] + x[n]\, h[9]$	$g[j] = h[j] + \sum_{m,i} x[n-m]\, h[i]$
	3	1	10	$g[3] \mathrel{+}= x[n-1]\, h[10]$	
	3	2	11	$g[3] \mathrel{+}= x[n-2]\, h[11]$	
	4	1	12	$g[4] \leftarrow h[4] + x[n-1]\, h[12]$	
	4	2	13	$g[4] \mathrel{+}= x[n-2]\, h[13]$	
	⋮	⋮	⋮	⋮	
2	0	0	3	$g[0] \leftarrow h[0] + x[n]\, g[3]$	$g[j] = h[j] + \sum_{m,i} x[n-m]\, g[i]$
	0	1	4	$g[0] \mathrel{+}= x[n-1]\, g[4]$	
	0	2	5	$g[0] \mathrel{+}= x[n-2]\, g[5]$	
	⋮	⋮	⋮	⋮	
1	–	0	0	$y[n] \leftarrow x[n]\, g[0]$	$y[n] = \sum_{m} x[n-m]\, g[m]$
	–	1	1	$y[n] \mathrel{+}= x[n-1]\, g[1]$	
	–	2	2	$y[n] \mathrel{+}= x[n-2]\, g[2]$	

Conclusion

- **Always faster than direct computation**
- **Trade-off between speed and flexibility**
- **Implementation in C shows a speedup up to 5**

4. Evaluation of Runtime

- We implemented 16 algorithms in C (4 methods of traversing $\times$ 4 methods of computation).
- We swept over order and memory with the number of parameters within $\{3, 9, 19, 34, 55, 83, 119, 164, 219, 285, 363, 454\}$.
- The source code is available at `www.spsc.tugraz.at/tools/fast-time-domain-volterra-filtering`.



Signal Processing & Speech Communication Lab
Graz University of Technology
Inffeldgasse 16c, A-8010 Graz, Austria
http://www.spsc.tugraz.at

The research leading to these results has received funding from the FFG Competence Headquarter program under the project number 4718971.