# An Architecture for Non-Linear Noise Filtering Via Piecewise Linear Compression

Konstantinos Konstantinides
Computer Peripherals Laboratory
Balas K. Natarajan
Computer Systems Laboratory
HPL-94-87
September, 1994

VLSI, signal processing

We present an architecture for non-linear noise filtering using a compression algorithm based on piecewise linear approximation. In contrast to spectral filters, the proposed technique does not require a priori knowledge of the noise and signal characteristics and the processing requirements are independent of its noise rejection properties. The architecture includes two multiplier-accumulator units, an adder, registers, and a short look-up table. The proposed implementation allows an output sample to be generated every four cycles on average.

Internal Accession Date Only

## 1. Introduction

Efficient filtering of signals corrupted with additive noise is central in signal processing. Traditional spectral filtering techniques [1] are effective in many cases, but often require some prior knowledge of the noise and signal characteristics. Furthermore, the processing requirements of spectral filters strongly depend on their noise rejection properties.

Recently, Natarajan [2] suggested a technique for reducing random additive noise from signals using data compression. The technique exploits the well known property that random noise is hard to compress, and does not require any prior knowledge of the signal or noise characteristics. In principle any compression technique can be applied for noise reduction. Piecewise linear compression has been shown to be particularly effective for this type of filtering [3]. In this paper we use a simplified version of an optimal piecewise linear approximation algorithm. The algorithm is better suited for hardware implementation without compromising on performance, and the complexity of the algorithm is independent of its noise rejection properties.

In Section 2 we review the principles of compression based filtering and we present a filtering technique based on piecewise linear approximation. The single-chip architecture is presented in Section 3. It includes two multiply-accumulate units, a register file, an adder, and a short look-up ROM table. The proposed implementation allows an output sample to be generated every four cycles on average.

## 2. Compression-Based Filtering

Consider the received data sequence $x(n) = s(n) + w(n)$, where $s(n)$ is an unknown signal and $w(n)$ is a random variable denoting noise of strength $b$ in a predetermined metric. In a recent paper, Natarajan [4] presented general theorems on learning functions in the presence of noise, which translate directly into theorems on noise filtering via data compression. The essence of these theorems is the following: Let $D$ be a lossy data compression algorithm that operates on $\{x(n)\}$ and a given tolerance $\epsilon$, and guarantees that its output $\{g(n)\}$ represents the input samples within $\epsilon$ in the above mentioned metric. Then, if $\epsilon$ equals the noise strength $b$, the noise and the signal loss cancel, with the extent of cancellation depending on the sampling rate and the effectiveness of the data compression routine.

This result leads to the following filtering technique: Compress the noisy signal with a a lossy data compression algorithm and set the allowed loss to be equal to the strength of the noise. The decompressed signal is the filtered output. This technique is rather general. Threshold filters using wavelet decompositions [5], and filters based on orthonormal bases and the minimum description length principle [6] can be viewed as special cases.

Since the strength of the noise $b$ is usually unknown, the following simple algorithm estimates an approximate value in a calibration step and then filters the data.

**input** $S = \{x(n)\}$
**Begin**
**Calibration phase**

Let $S_1$ be a representative sample of $S$.
Compress $S_1$ with $D$ for various values of the loss tolerance $\epsilon$.
Plot compressed size as a function of $\epsilon$.
Let $\hat{\epsilon}$ be the knee-point of the plot.

**Filtering phase**

Compress $S$ with $D$ using $\epsilon = \hat{\epsilon}$ to get $\{g(n)\}$.
Decompress $\{g(n)\}$ to obtain $\{y(n)\}$, the filtered signal.

**End**

If $s$ denotes the compressed size of the input signal, the knee-point is defined as the point at which the second derivative $d^2s/dlog(\epsilon)^2$ is maximum.

While any lossy compression scheme could be used, in this paper we apply the above filtering technique using as $D$ a compression algorithm based on the piecewise linear approximation (PLA) of waveforms with respect to the $L_\infty$ metric. The essence of the PLA problem is that given a piecewise linear function $x:[a,b]\rightarrow R$ with $N$ sample points and tolerance $\epsilon \in R$, to construct a piecewise linear function $g$ such that for all $t \in [a,b]$, $|x(t)-g(t)| \leq \epsilon$ and $g$ consists of the fewest number of segments over all such functions.

There are many algorithms for the piecewise linear approximation of waveforms [7]. Imai and Iri [8] have developed an optimal algorithm that runs in time linear on the number of samples. However, its arithmetic complexity resists efficient design of a custom VLSI implementation. Most recently [9], we presented a simplified version of that algorithm. The number of operations on the modified algorithm is substantially smaller than those of the optimal algorithm, and for uniformly sampled data, the modified algorithm is particularly attractive for a single-chip hardware implementation. The modified algorithm is not optimal, but in practice, we find that the average number of segments constructed by the modified algorithm is roughly 1.5 of the optimum number, and that in filtering applications it only performs slightly worse than the optimal one [3].

For uniformly sampled data, a description of the compression/filtering algorithm is given in Fig. 1. The essence of this algorithm is to construct a tunnel of radius $\epsilon$ around the original data and to find, starting from the start of the tunnel, the fewest line segments that span the tunnel (see Fig. 2).

**input:** S=$\{x(n)\}$, $\epsilon$, compression/filter flag.
**Begin**
Let $x^+(n) = x(n) + \epsilon$ , $x^-(n) = x(n) - \epsilon$
$x_s = x(1), T = 1, p = n = 1, y(0) = x(1)$;
**while** S is not empty **do**
$T = T + 1$, $n = n + 1$;
$a_h = x^+(n) - x_s$, $a_l = x^-(n) - x_s$;
$T = T + 1$, $n = n + 1$;
**while** $x^+(n) \geq a_l T + x_s$ **and** $x^-(n) \leq a_h T + x_s$
**do**
**if** $x^+(n) < a_h T + x_s$
**then** $a_h = (x^+(n) - x_s)/T$;
**if** $x^-(n) > a_l T + x_s$
**then** $a_l = (x^-(n) - x_s)/T$;
$T = T + 1$, $n = n + 1$;
**end**
**if** $x^+(n) < a_l T + x_s$ **then** $a_s = a_l$;
**else** $a_s = a_h$;
**if** *compression* **then**
$g(n-1) = a_s(T-1) + x_s$;
**elseif** *filtering* **then**
**for** $k = p + 1$ to $k = n - 1$ **do**
$y(k) = y(k-1) + a_s$;
$x_s = x(T-1)$, $T = 0$, $p = n = n-1$, $y(n) = x_s$;
**end**
**End**

Fig. 1: Algorithm for piecewise linear compression and filtering.

Starting from $x_s = x(1)$, the algorithm tries to draw tangents to increasingly longer prefixes of the upper and lower envelopes. Specifically, initially the upper and lower envelopes are truncated to the first two points. The algorithm then draws a tangent to each of the envelopes (with slopes $a_h$ and $a_l$) from the point $x_s$. It then seeks to extend each envelope to the next point, and update their tangents to include them. This is carried on iteratively until no tangent exists when the next point is included. The algorithm then outputs the longest tangent as its first line segment, and cuts off the portion of the envelopes examined so far. The entire process is then repeated on the remaining portion of the tunnel to obtain the second line segment, and so on. Decompression is performed by simple linear interpolation.

### 2.1 Example

Consider $N = 1000$ samples of the waveform given by $x(n) = s(n) + w(n)$, $n = 1, 2, \ldots, N$, where $s(n) = \sin\left(\frac{1}{n/N + 0.01}\right)$, and $w(n)$ is uniformly distributed noise in the

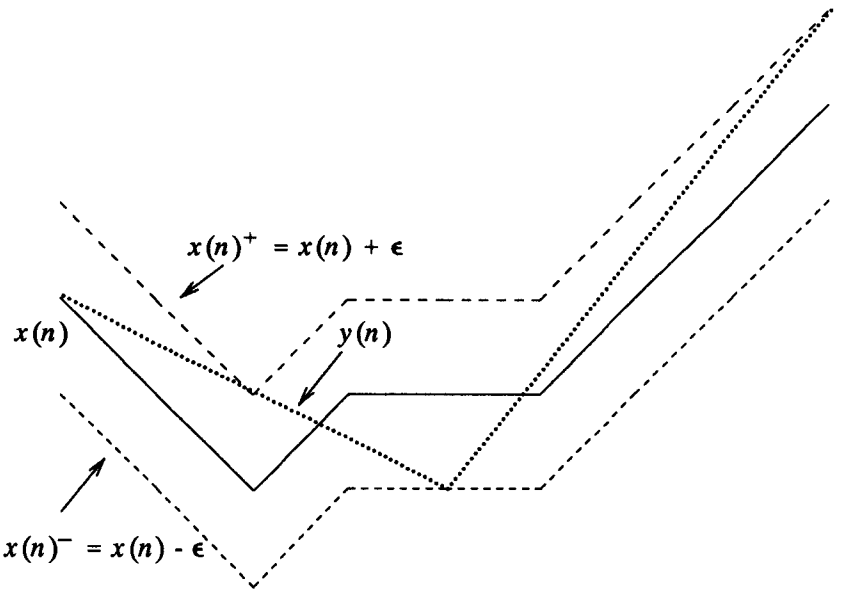


Fig. 2: Piecewise linear filtering.

interval [-0.1, 0.1]. Fig. 3 is a plot of the compressed size of the data for various values of $\epsilon$ using the compression algorithm of Fig. 1. The second derivative of the plot is also shown. From Fig. 3, the knee-point is selected as $\hat{\epsilon} = 0.11$ to be the value of $\epsilon$ at which the second derivative plot attains a maximum. Fig. 4 shows the frequency spectrums of the noise-free signal and the filtered signal $\{y(n)\}$ using $\epsilon = 0.11$. For comparison, it also shows the spectrum of the output signal from a 6-th order low-pass Butterworth filter with cut-off frequency $f_c = 0.2$. From Fig. 4, the output spectrum of the compression filter matches much better than the Butterworth filter the original spectrum. As expected, the Butterworth filter removes all the high-frequency components of the original signal.

## 3. Architecture

From Fig. 1, the main computations can be summarized as follows:
a) Tunnel Generation

$$x^+(n) = x(n) + \epsilon \ , x^-(n) = x(n) - \epsilon \tag{1a}$$

b) Tangent Generation

$$d^+(n) = x^+(n) - x_s \ , \ d^-(n) = x^-(n) - x_s$$

$$\hat{a}_h = d^+(n)/T \ , \ \hat{a}_l = d^-(n)/T \tag{1b}$$

c) Loop Termination Tests

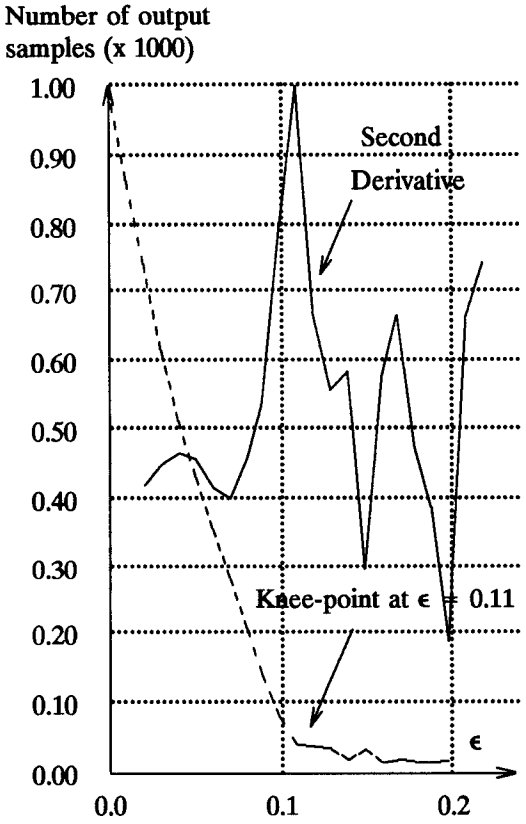


Fig. 3: Filtered output versus $\epsilon$, and its second derivative.

$$S_1(n) = d^+(n) - a_l T \, , \; S_2(n) = d^-(n) - a_h T \tag{1c}$$

d) Tangent Update Tests

$$S_3(n) = d^+(n) - a_h T \, , \; S_4(n) = d^-(n) - a_l T \tag{1d}$$

e) Output Generation (Interpolation)

$$y(k) = y(k\text{-}1) + a_s \tag{1e}$$

From Fig. 1, if $S_1(n) < 0$ or $S_2(n) > 0$ then the main while-loop terminates and the algorithm starts the output data generation. Similarly, if $S_3(n) < 0$ or $S_4(n) > 0$, then new slopes ($\hat{a}_h$ and $\hat{a}_l$) are computed using (1b).

From (1), the filtering algorithm requires nine unique additions and at most two multiplications and two divisions per sample point. In fact, the two divisions can easily be replaced by a table look-up and two multiplications. There are many ways to implement the above operations in hardware. A traditional programmable digital signal processor (DSP) would not be very efficient since the algorithm does not have a

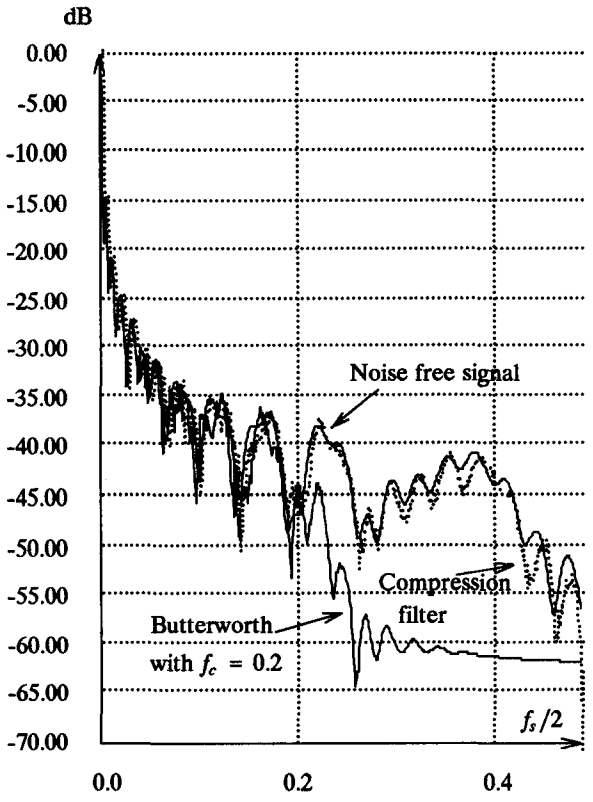


Fig. 4: Frequency spectrum of the noise-free signal, the output of the compression-based filter, and the output of a 6-th order Low-pass Butterworth filter.

continuous stream of multiply-accumulate operations for which traditional DSPs are so efficient.

Fig. 5 shows a block diagram of a hardware implementation of the algorithm that provides a good balance between hardware complexity and efficiency of implementation. It includes a controller, a small set of registers for local storage of temporary data, a time counter, two multipliers, three adders, and a ROM look-up table. The size of the ROM table depends on the maximum value of $T$. Since $T$ can never be larger than the length of the maximum segment that spans a part of the data tunnel for a given $\epsilon$, it also represents the maximum compression ratio achieved for the tolerance $\epsilon$. In most applications, the average maximum compression ratio is less than 20 to 1.

Let $a_h^*$ and $a_l^*$ denote the values of the slopes of the upper and lower tangents used in a particular stage of the algorithm. Let $P_h = a_h^* T$ and $P_l = a_l^* T$. Table 1 shows the pipeline flow of operations for equations (1b-1d) using the proposed architecture. From Table 1, the main while loop of the algorithm can be evaluated in three clock cycles per input sample. The updated slopes, $\hat{a}_h$ and $\hat{a}_l$, are computed for each sample point, regardless of whether they are needed or not. The slopes $a_l^*$ and $a_h^*$ are replaced with their updated values only if either test $S_3$ or $S_4$ fails. For example, from Table 1,

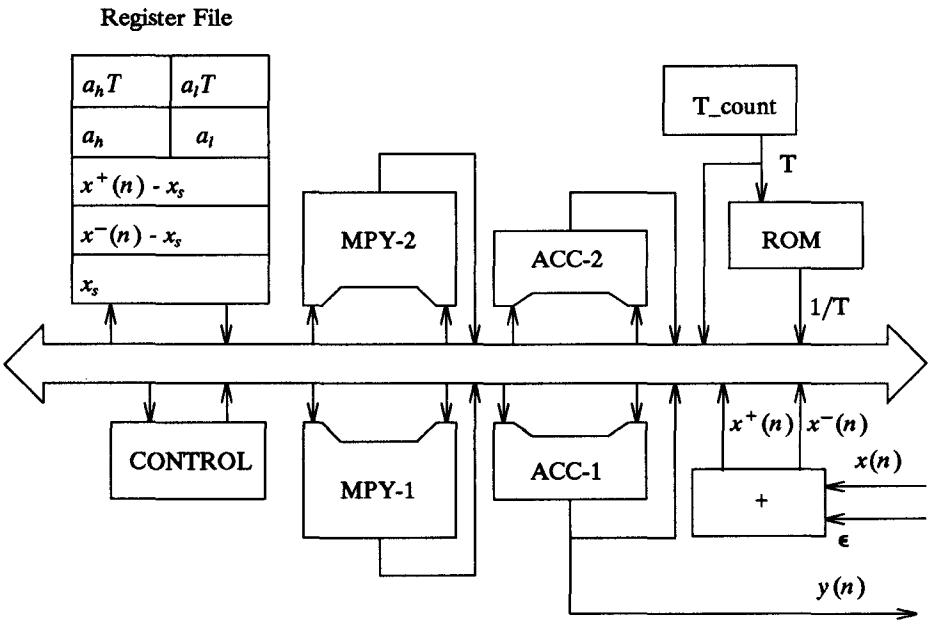


Fig. 5: Block diagram of the compression based filter architecture.

Table 1
Pipelined Data Flow in Filtering Algorithm

| | Clock | | | |
|---|---|---|---|---|
| Inputs | t | t+1 | t+2 | t+3 |
| ACC-1 | $x^+(n) - x_s$ | $d^+(n) - P_h$ | $d^+(n) - P_l$ | $x^+(n+1) - x_s$ |
| ACC-2 | $x^-(n) - x_s$ | $d^-(n) - P_l$ | $d^-(n) - P_h$ | $x^-(n+1) - x_s$ |
| MPY-1 | $a_l^* T$ | $d^+(n)\, T^{-1}$ | | $a_l^* T$ |
| MPY-2 | $a_h^* T$ | $d^-(n)\, T^{-1}$ | | $a_h^* T$ |
| Outputs | | | $T = T + 1$ | |
| ACC-1 | $S_1(n-1)$ | $d^+(n)$ | $S_3(n)$ | $S_1(n)$ |
| ACC-2 | $S_2(n-1)$ | $d^-(n)$ | $S_4(n)$ | $S_2(n)$ |
| MPY-1 | | $P_l$ | $\hat{a}_h$ | |
| MPY-2 | | $P_h$ | $\hat{a}_l$ | |

if at time $t+2$ $S_3(n) < 0$ then at $t+3$ $a_h^* = \hat{a}_h$. If the evaluation of $S_1$ or $S_2$ flags a loop termination, then the computed values in the pipeline are discarded and one of the adders is used to evaluate in a pipelined mode the output data samples using

$$y(i) = y(i\text{-}1) + a_s \ , \ i = 1, 2, ..., T\text{-}1 \qquad (2)$$

where $y(0) = x_s$, $a_s = a_l^*$ if $S_1 < 0$ and $a_s = a_h^*$ otherwise.

## 4. Conclusions

Compression based filtering can be applied in a variety of signal processing problems. It is particularly effective in filtering broad-band signals and when there is no prior knowledge of either the signal or noise characteristics. We presented an architecture for noise filtering using a fast and efficient algorithm for piecewise linear approximation. In pipelined mode, the design requires three cycles per input sample for the compression phase of the algorithm and one cycle per sample for the decompression phase.

### References


1. S. A. Tretter, "*Introduction to discrete-time signal processing,*" Wiley, 1976.
2. B. K. Natarajan, "Filtering random noise via data compression," Proc. IEEE Data Compression Conf., Snowbird, Utah, 1993, pp. 60-69.
3. B. K. Natarajan and K. Konstantinides, "Comparing Occam and Wiener filters on broad-band signals," to appear in Proc. of 1994 Asilomar Conference on Signals, Systems, and Computers, Nov. 1994.
4. B. K. Natarajan, "Occam's razor for functions," Proc. ACM Conf. on Comp. Learning Theory, Santa Cruz, CA, Jul. 1993.
5. D. L. Donoho, I.M. Johnstone, G. Kerkyacharian, and D. Picard, "Wavelet Shrinkage: Asymptopia?," Tech. Rep. 419, Dept. of Statistics, Stanford University, Stanford, CA, 1993.
6. N. Saito, "Simultaneous noise suppression and signal compression using a library of orthonormal bases and the minimum description length principle", Wavelets in Geophysics, to appear in 1994.
7. G. Papakonstantinou, P. Tsanakas, and G. Manis, "Parallel approaches to piecewise linear approximation," Signal Processing, Vol. 37, pp. 415-423, Elsevier Science B. V., 1994.
8. H. Imai and M. Iri, "An optimal algorithm for approximating a piecewise linear function," Journal of Information Processing, Vol. 9, No. 3, pp. 159-161, 1986.
9. K. Konstantinides and B. K. Natarajan, "An architecture for lossy compression," *VLSI Signal Processing V*, K. Yao et al. Editors, IEEE, 1992, pp. 237-246.