A Parallel Programmable Architecture for Linear and Nonlinear Filtering

Lori Lucke
Dept. of Electrical Engineering
University of Minnesota
Minneapolis, MN 55455, USA
lucke@ee.umn.edu

Chaitali Chakrabarti
Dept. of Electrical Engineering
Arizona State University
Tempe, AZ 85287-5706, USA
chaitali@enuxsa.eas.asu.edu

ABSTRACT — **In this paper we introduce a parallel, programmable architecture capable of supporting both linear and nonlinear filtering. The proposed architecture consists of a ring of processors which can be configured to operate like a sorter for order statistic based filters, and like a systolic array for linear filters. Such a structure also supports combination filters that are obtained by cascading linear and non-linear flters. In addition, this architecture efficiently computes filters with window sizes larger than the number of processors.**

1 Introduction

In digital signal and image processing applications, removal of noise is achieved by nonlinear filters, linear filters, and a combination of these two filters. Each of these filters are effective in removing certain types of noise distributions. For instance, linear filters effectively remove Gaussian noise. Nonlinear filters, based on order statistics, effectively remove impulsive noise and noise with a heavy-tailed distribution. In the past, several application-specific architectures have been proposed to implement linear and nonlinear filters. The limited applicability of these architectures makes the study of programmable structures which can support both linear and nonlinear filters very important. In this paper, we introduce a new programmable architecture that is capable of implementing nonlinear filters based on order statistics as well as linear FIR filters and combination filters.

Many nonlinear filters such as the rank-order filters, weighted order statistic (WOS) filters, stack filters, morphological filters, etc. are based on order statistics [1]. For an order statistics filter, the output is equal to one of the sample values in the input window and depends only on the rank and position of the sample value. Combination filters are those filters which attempt to combine the performance of an order statistic filter with a linear filter [1, 2]. The simplest combination filter is the α-trimmed mean filter. This simple filter can be extended to the class of L-filters [1], where the sample window is ordered (before filtering) and then combined using linear techniques similar to the FIR filter. There are other combination filters where linear filtering is done prior to ordering, such as the FIR-WOS hybrid filters [3]. In all these filters, the output is calculated using a local neighborhood or window of sample points. The window of sample points may be one or multi-dimensional. The window slides over the input data and an output is calculated for each window position.

There exist several application-specific architectures [1, 4, 5, 6] as well as general-purpose architectures [7, 8] for both linear and order statistic filters. However there are only a few general purpose implementations which support *both* linear and nonlinear filters [7, 9]. The architecture in [7] consists of a programmable sorter which sorts windows of size 5 and 7 with two compare-swap units, and variable delay registers to implement sliding average type linear filters. The architecture in [9], on the other hand, consists of a single multiplier and a small parallel programmable rank order filter block.

In this paper we present an architecture consisting of a ring of processors, which can be configured to operate like a sorter for order statistic based filtering and like a systolic array for linear filtering. Such a structure enables efficient cascading of linear and non-linear operations, making it highly suitable for combination filtering. In addition, operations where the window size is larger than the number of processors, can be efficiently mapped into this architecture with little overhead.

2 A Programmable Architecture for Nonlinear/Linear Filters

We propose a new parallel, programmable architecture for linear and nonlinear filtering. This architecture consists of a circular ring of N processors, $P(0)$ to $P(N-1)$, controlled by a sequencer as shown in Fig. 1. Each processor contains register files, a comparator, a multiplier, an adder, and control logic as shown in Fig. 2. With this configuration it is possible to support all the functions necessary for order statistic filters, FIR filters, and combination filters.

Processor Design: The data path of a processor consists of a comparator, a multiplier, and an adder. The comparator is used for implementing sorting operations in nonlinear filtering. The comparator compares an input from the input register file with an input from the neighboring processor (left or right). The result of this comparison causes the inputs to be

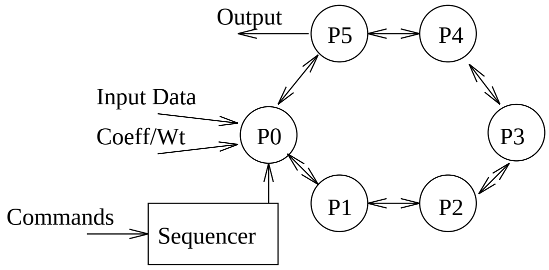

Figure 1: Ring of $N = 6$ Processors

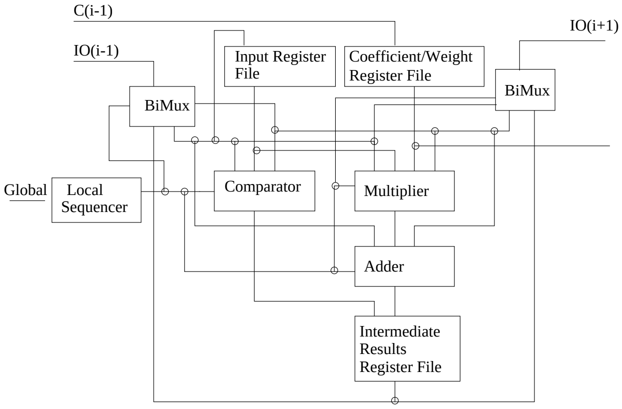

Figure 2: Block Diagram of a Processor

swapped (and also the weights to be swapped in a WOS application) as needed. The multiplier supports FIR filtering operations by multiplying an input from the register file with a coefficient from the coefficient file. The adder supports both FIR filtering operations and WOS filtering operations. The adder accumulates the results of the multiplication in an FIR operation by adding the current result with the result from the processor to the left. The adder also supports WOS operations by accumulating the weights of the current processor with the processor to the left. The global sequencer supports the microcode necessary to control each operation.

There are three register files in each processor. The register files provide support for operations where the window size, M, is larger than the number of processors, N. Let $M = KN$. Then the input register file holds up to K input samples. The input registers in processor $P(j)$ are labeled $IR_i(j)$, where $0 \leq i < K$. The coefficient/weight register file holds the corresponding input coefficients for an FIR filtering operation or the corresponding weights for a WOS filtering operation. The coefficient file is also of size K and the coefficient registers are labeled $CR_i(j)$, where $0 \leq i < K$. The output registers are labeled $OR_i(j)$, and hold intermediate results, where $0 \leq i < K$.

In the next section we demonstrate how this architecture can be used to implement a rank order filter, a weighted order statistic filter, an FIR filter, an α-trimmed mean filter, and an L-filter. Other combination filters can be implemented with this structure. We omit those descriptions for lack of space.

3 Implementation of Nonlinear/Linear Filters

3.1 Rank Order Filter

A rank order filter output is calculated by ordering the inputs in the sample window and choosing the element with a particular rank as the output. Such a filter is implemented by configuring the processor ring to function as a bubble sorter. We chose to implement the sorter with a bubble sort instead of a running order sort because of its simplicity and ability to handle multi-dimensional windows more efficiently. In this implementation, alternating pairs of processors compare and swap the input data. The sort requires M steps where M is the number of inputs in the sample window. The sorting operation requires a compare-swap suboperation, CSW, to be computed in each processor. CSW compares an input from the current input register file in processor $P(i)$ with an input from the processor to the right, $P(i+1)$. A CSW operation executed in processor $P(N-1)$ performs the comparison with processor $P(0)$. The inputs are swapped, if necessary, and returned to their respective processors.

We next describe the sequence of operations when the window size, M, is larger than the number of processors, N. For simplicity we assume M and N are even, M is a multiple of N, and $K = M/N$. In this case, each processor holds K inputs. The rank order filter is performed using a bubble sort. Thus M compare and swap steps are necessary to sort the entire input set with $M/2$ comparisons occurring at even steps on even processors; and $M/2 - 1$ comparisons occurring at odd steps on odd processors. Since $M = KN$, this step is decomposed into K substeps with $N/2$ or $N/2 - 1$ comparisons occurring at each substep.

Algorithm Rank Order Filter:

Step 0: Inputs: $I(k)$, $0 \leq k < M$

Step 1: Load inputs serially.
For $j = 0$ to $M - 1$ do $IR_{\lfloor j/N \rfloor}(j \bmod N) := I(j)$.

Step 2: Perform a bubble sort
Repeat Steps 2.1 and 2.2 $M/2$ times
2.1 Even compare and swap
For $i = 0$ to $K - 1$ Begin
CSW$(IR_i(2j), IR_i(2j+1))$, $0 \leq j \leq N/2$
Endfor
2.2 Odd compare and swap
(Note wraparound from $P(N)$ to $P(0)$)
For $i = 0$ to $K - 2$ Begin
CSW$(IR_i(2j+1), IR_i(2j+2))$,
for all $0 \leq j \leq N/2 - 2$
CSW$(IR_i(N-1), IR_{i+1}(0))$
Endfor
CSW$(IR_{K-1}(2j+1), IR_{K-1}(2j+2))$, for all
$0 \leq j \leq N/2 - 2$

Step 3: Output the input with rank $= R$.
Output $:= IR_{\lfloor R/N \rfloor}(R \bmod N)$

The proposed rank order filter algorithm sorts all the inputs in a window from scratch every time. Consequently, all the inputs have to be loaded afresh for every output computation. It is also possible to program this filter to support one-dimensional rank order filters, where a single input is shifted in for each new output. In such a case, the compare-swap operations are done on the output register file (except for the first which operates on the input file).

3.2 Weighted Order Statistic Filter

A weighted order statistic filter is a rank order filter where each input has a weight associated with it. The output can be calculated by sorting a repeated input set (each input is repeated by its weight value) and finding a particular ranked input in the (repeated and sorted) input set. Instead, we use the approach where the output is found by sorting the original input set and then adding the weights in order to determine the weighted rank of each input as in [5]. The WOS filter is implemented by configuring the processor ring to operate as a bubble sorter as above. Each weight is now carried along with the input as the inputs are sorted. Once sorted, the weights are summed in order, and compared to a threshold to determine the correctly ranked output. In addition to the CSW operation, the WOS filter requires a SWP and an ADD operation. The SWP operation swaps the corresponding input weights using the result of the previous CSW operation. The ADD operation adds the weights to determine the weighted ranks.

We next describe the sequence of operations for the case when $M > N$. For simplicity we assume M and N are even, M is a multiple of N and $K = M/N$. The other cases are similar.

Algorithm WOS Filter:

Step 0: Inputs: $I(k)$, $0 \leq k < M$
Weights: $W(k)$, $0 \leq k < M$

Step 1: Load inputs serially.
For $j = 0$ to $M-1$ Begin
$IR_{\lfloor j/N \rfloor}(j \bmod N) := I(j)$
$CR_{\lfloor j/N \rfloor}(j \bmod N) := W(j)$
Endfor

Step 2: Perform a bubble sort
Repeat Steps 2.1 and 2.2 $M/2$ times
2.1 For $i = 0$ to $K-1$ Begin
$\mathrm{CSW}(IR_i(2j), IR_i(2j+1))$ AND
$\mathrm{SWP}(CR_i(2j), CR_i(2j+1)$,
for all $0 \leq j < N/2$
Endfor
2.2 For $i = 0$ to $K-2$ Begin
$\mathrm{CSW}(IR_i(2j+1), IR_i(2j+2))$ AND
$\mathrm{SWP}(CR_i(2j+1), CR_i(2j+2)$,
for all $0 \leq j \leq N/2 - 2$
$\mathrm{CSW}(IR_i(N-1), IR_{i+1}(0))$ AND
$\mathrm{SWP}(CR_i(N-1), CR_{i+1}(0)$
Endfor
$\mathrm{CSW}(IR_{K-1}(2j+1), IR_{K-1}(2j+2))$ AND
$\mathrm{SWP}(CR_{K-1}(2j+1), CR_{K-1}(2j+2))$, for all
$0 \leq j \leq N/2 - 2$

Step 3: Accumulate the weights
For $i = 0$ to $K-1$ Begin
For $j = 1$ to $N-1$ Begin
$CR_i(j) := \mathrm{ADD}(CR_i(j-1), CR_i(j))$
Endfor
If $i < K-1$ Then
$CR_{i+1}(0) := \mathrm{ADD}(CR_i(N), CR_{i+1}(0))$
Endfor

Step 4: Output the input with weighted rank $= R$
For $j = 0$ to $N-1$ Begin
For $i = 0$ to $K-1$ Begin
If $CR_i(j) == R$, Output $= IR_i(j)$
Endfor
Endfor

3.3 FIR Filter

A one-dimensional FIR filter can be implemented on the processor ring by moving the inputs and the partial results in opposite directions while the coefficients stay in the processors [10]. In this method, the inputs are interspersed with 0s and valid computations are done in each processor every other cycle. The first output is generated $2M-1$ cycles after the first input is loaded. After that, the outputs come out every other cycle from processor $P(0)$. FIR filters can be implemented using a systolic array of processors by several other methods as illustrated in [10].

When the order of the filter, M, is larger than the number of processors N, the above implementation can easily be folded. Let M be a multiple of N, $K = M/N$. Then each processor stores K coefficients, and computes K partial results in each step. The inputs are loaded through IR_0 of processor $P(0)$, while the computations for each output start in processor $P(N-1)$. The partial results and the inputs are circulated K times through the array and the outputs are read out from processor $P(0)$.

Algorithm FIR Filter:

Step 0: Load coefficients and initialize partial results.

Coefficients: $C(k)$, $0 \leq k < M$
For $j = 0$ to $M-1$ Begin
$CR_{\lfloor j/N \rfloor}(j \bmod N) := C(j)$
$OR_{\lfloor j/N \rfloor}(j \bmod N) := 0$
Endfor

Step 1: Repeat in each cycle.
1.1 Load external input in $IR_0(0)$.
Set $OR_{K-1}(N-1)$ to 0.
1.2 For $i = 0$ to $K-1$ Begin
$OR_i(j) := OR_i(j) + IR_i(j) * CR_i(j)$
$IR_i(j)$ shifted to $IR_i(j+1)$, for $0 \leq j < N-1$
$IR_i(N-1)$ shifted to $IR_{i+1}(0)$
$OR_i(j)$ shifted to $OR_i(j-1)$, for $1 < j \leq N-1$
$OR_i(0)$ shifted to $OR_{i-1}(N-1)$

Endfor

1.3 Output contents of $OR_0(0)$.
(Output will be valid only every other cycle.)

The above algorithm describes a one-dimensional systolic FIR filter. It is also possible to support multi-dimensional FIR filters with this architecture. In such a case, all the inputs corresponding to an output are loaded into the filter as in the WOS algorithm. The output is then be accumulated across the processor array.

3.4 α-Trimmed Mean Filter

The output of an α-trimmed mean filter is found by ordering the inputs, removing α outliers, and averaging over the remaining inputs.

$$y_i = \frac{1}{M(1-2\alpha)} \sum_{j=\alpha M+1}^{M-\alpha M} x_{(j)}$$

where $x_{(j)}$ is the element with rank j.

An α-trimmed mean filter can be implemented on the processor ring by first sorting the inputs using bubble sort, and then adding the sorted inputs whose ranks lie between $\alpha M + 1$ and $M - \alpha M$. The bubble sort procedure is the same as described in Section 3.1. At the end of the sorting procedure, the sorted input lies in the output register files of the processors. Each processor then computes the sum of all the inputs (residing in its output register file) whose ranks lie between $\alpha M + 1$ and $M - \alpha M$. The partial sums generated in each processor are then added across the array. The final sum is multiplied by the constant $\frac{1}{n(1-2\alpha)}$ in processor $P(N-1)$ and the result is output from $P(N-1)$.

Algorithm α-Trimmed Mean Filter:

Step 1: Load inputs serially.

For $j = 0$ to $M-1$ do $IR_{\lfloor j/N \rfloor}(j \bmod N) := I(j)$.

Load multiplier coefficient
$CR_0(N-1) = \frac{1}{M(1-2\alpha)}$

Step 2: Perform bubble sort as in Sec. 3.1.

Step 3: Accumulate the inputs in each processor.

for $j = 0$ to $N-1$ Begin
$IR_0(j) = 0$
For $h = \alpha M + 1$ to $M - \alpha M$ Begin
if (j= $h \bmod N$) then
i := $\lfloor h/N \rfloor$
$IR_0(j) := IR_0(j) + OR_i(j)$
Endfor
Endfor

Add the partial sums generated in each processor

For $j = 1$ to $N-1$ do
$IR_0(j) := IR_0(j) + IR_0(j-1)$

Step 4: Generate output.

Output = $CR_0(N-1) * IR_0(N-1)$

3.5 L filter

An important generalization of the α-trimmed mean filter is the L-filter.

$$y_i = \sum_{j=1}^{n} a_j x_{(j)}$$

where a_j are the filter coefficients and $x_{(j)}$ is the element with rank j. The L-filter can be implemented by cascading a bubble sort with an FIR filter. The procedure is very similar to the α-trimmed mean filter. The main difference is that each processor now multiplies the contents of its output registers, OR, with the weights stored in its CR registers, before accumulating. The partial sums generated in each processor are then added as in α-trimmed mean filters.

4 Implementation

We are currently implementing the programmable filter using Mentor Graphics CAD tools and a 1.2 um standard cell library for the case where $N = 4$ and $K = 2$.

References

[1] I. Pitas and A.N. Venetanopoulos. *Nonlinear Digital Filters*. Boston: Kluwer, 1989.

[2] Y. Neuvo, P. Heinonen, and I. Defee. Linear - median hybrid edge detectors. *IEEE Trans. on Circuits and Sys.*, CAS-34(11):1337–1343, November 1987.

[3] L. Yin and Y. Neuvo. Fast adaptation and performance characteristics of FIR-WOS hybrid filters. *IEEE Trans. on Signal Processing*, pages 1610–1628, July 1994.

[4] C. Chakrabarti and L. Lucke. Highly concurrent architectures for recursive median filters. In *VLSI Signal Processing, VI*, pages 471–479. IEEE Press, 1993.

[5] L.E. Lucke and K.K. Parhi. VLSI architectures for weighted order statistics filters. In *Proc. of the IEEE Winter Workshop on Nonlinear Digital Signal Processing*, January 1993.

[6] G. Casagrande and et. al. VLSI programmable digital filter for video signal processing. *Journal of VLSI Signal Processing*, pages 219–232, December 1993.

[7] O. Vainio, Y. Neuvo, and S.E. Butner. A signal processor for median-based algorithms. *IEEE Trans. on Acous., Speech, and Sig. Proc.*, September 1989.

[8] N.J. Leite and M. de Barros. A highly reconfigurable neighborhood image processor based on functional programming. In *Proc. IEEE Int'l Conf. on Image Proc.*, November 1994.

[9] J. Nurmi and et. al. A nonlinear DSP core architecture. In *Proc. IEEE Winter Workshop on Nonlinear Digital Signal Processing*, January 1993.

[10] H. T. Kung. Why systolic architectures ? *IEEE Computer*, pages 37–46, January 1982.