Fast Running Max/Min Filters

Dinu Coltuc, Ioannis Pitas
Department of Electrical & Computers Engineering
University of Thessaloniki, Thessaloniki 540 06, GREECE
email: {dinu,pitas}@vergina.eng.auth.gr

ABSTRACT — **The paper focuses on the fast computation of max/min running filters. For arbitrary n (window filter size) the existence of a fast algorithm of $O(\log_2 n)$ comparisons per sample complexity is proven. A remarkable feature of the proof is its ability to generate a particular solution for each n. The implementation of fully-programmable pipeline processors for variable size window is analyzed.**

1 Introduction

Max/min filters are well-known and widely used in nonlinear signal processing applications [1]-[3]. Their computational complexity, i.e. the number of comparisons per sample, depends on the size of the filter window. Direct evaluation leads to a computational complexity of $O(n)$ comparisons per sample. The particular case when the size of the filter window is a power of two has already been investigated in the literature and fast implementations of the order of $O(\log_2 n)$ have been presented [4]. The basic idea is to recursively divide the sequence of n samples in subsequences of $n/2$ samples, and so on, until subsequences of size two are obtained. The maximum (minimum) of each subsequence is further used in two comparisons and, obviously, it is computed only once. The demand that n be a power of two assures a perfect decomposition of the computation in $\log_2 n$ steps.

This paper extends the $O(\log_2 n)$ complexity max/min algorithm for arbitrary size window. Our approach is based on the algebraic properties of the max/min operators, namely associativity, commutativity and especially idempotence and not on the arithmetic properties of n, the window size. A computational structure is proposed and the validity of the result is proven. The proposed structure leads to a pipeline implementation.

2 Fast computation

Since maximum and minimum are dual operations, if one is implemented, the other is obtained by reverting the data before and after the computation. Therefore, without any loss of generality, we further refer only to the maximum computation. Let X, where $X(i) = x_i$ for $i = 0, \ldots, N$, be a given input sequence. The problem of 1D running max filter within a window of size n is to determine a sequence Y, $Y(i) = y_i$, where y_i is the maximum of n consecutive samples:

$$y_i = \max(x_i, x_{i-1}, \ldots, x_{i-n+1}) \qquad (1)$$

Let D be the delay operator, $D^m(X)(i) = X(i-m)$. We have $D^m(D^n(X)) = D^n(D^m(X)) = D^{m+n}(X)$ We assume that $X(i) = 0$ for $i < 0$. Using the delay operator, the output of the running max filter on X is:

$$Y(i) = \max(D^0(X), \ldots, D^{n-1}(X))(i) \qquad (2)$$

It is easy to see that in the evaluation of any two consecutive elements, e.g. y_i and y_{i+1}, all the operands but two are common:

$$y_i = \max(\max(x_i, x_{i-1} \ldots, x_{i-n+2}), x_{i-n+1}) \quad (3)$$

$$y_{i+1} = \max(x_{i+1}, \max(x_i, x_{i-i}, \ldots, x_{i-n+2})) \quad (4)$$

This observation is of great importance in reducing the computational complexity of the running filter. The idea is to find a factorization of the computation that evaluates only once the common terms involved in (3), (4).

We further introduce a flow graph (Fig. 1) to describe the computation of the above filters. Its basic elements are nodes connected by branches. The branches are labeled by delay operators. Only one delay operator labels each branch. The sequence transmitted by each branch is delayed by its corresponding delay operator. For simplicity reasons, we omit to label the branches delayed

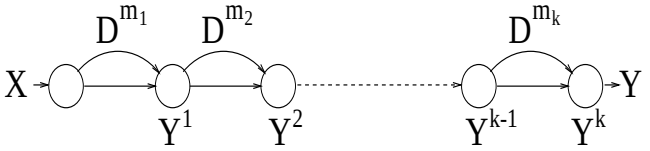

Figure 1: General structure; data flow graph.

by D^0, the identity delay operator. The nodes are places where maximum among the operands of the incoming branches is computed. They may be labeled with the result of the computation. No computation takes place in the input node.

In the sequel, we prove that, for any window size n, there are factorizations of the computation that lead to a computational complexity of order $O(\lceil \log_2 n \rceil)$, where $\lceil x \rceil$ is the least integer greater than x. Our approach considers a general computational structure (Fig. 1) that is constrained to compute the desired filter output by specifying its elements, i.e. the number of nodes k and the values of delay constants D^{m_i}. By labeling the nodes j with the result of the computation Y^j, we have the following equation:

$$Y^j(i) = \max(Y^{j-1}, D^{m_j}(Y^{j-1}))(i) \qquad (5)$$

where $j = 1, 2, \ldots k$. The final result of the structure may be derived either by recursive substitution or by direct evaluation. Each node performs a max operation between the input and the delayed input. Thus, each node multiplies by two the number of operands involved in the computation. For the final node k, there are 2^k operands, i.e. the input sequence X and the sequences obtained by all the possible combinations of the delay operators $D^{m_1}, D^{m_2}, \ldots, D^{m_k}$. Maximum operation is commutative, thus, the order of the operands may be changed with no effect on the result. Let I be the set of delay values, $I = \{m_1, m_2, \ldots, m_k\}$, and let $\mathcal{P}(I)$ be the power set of I. For each $I_j \in \mathcal{P}(I)$ we denote by $S(j)$ the sum of the elements of I_j. With the above notation, the output of the structure is:

$$Y(i) = \max(D^{S(0)}(X), \ldots, D^{S(2^k-1)}(X))(i) \quad (6)$$

We want to compute the running max filter by using the structure presented in Fig. 1. To do this, we have to determine under what conditions equations (2) and (6) give the same results.

Notice that for n power of two, i.e. $n = 2^k$, the right side of (2) and (6) have the same number of operands. The delay factors $I = \{m_1, m_2, \ldots, m_k\}$ have to be determined so that the operands in both equations are identical. The set $I = \{1, 2, \ldots, 2^{k-1}\}$ is a solution, since, for $j = 0, \ldots, 2^k - 1$, $S(j)$ takes all the values in $[0, n-1]$. The k-bits binary representation of any integer m, $0 \leq m < 2^k - 1$, gives an immediate proof of this assertion. The above solution recovers the structure proposed in [4]. The computation of running max for $n = 2^k$ is done only in k operations, $k = \log_2 n$.

Let us consider the general case when n is an arbitrary integer. Since the number of operands in the right-hand side of equations (2) and (6) is different, the idempotence property of maximum is of crucial importance. Equation (6) gives the running max on X, if the following three conditions are met:

1. $2^k \geq n$, i.e., there are enough operands in (6).

2. $S(j) < n$, i.e., the delay operators in (6) do not exceed the range of delay operators in (2).

3. $S(j)$ takes all the values in the closed interval $[0, n-1]$, i.e., all the delay operators in (2) appear also in (6).

We shall prove that for each n there is a minimal set I that satisfies the above conditions. From condition 1, we have $k \geq \log_2 n$. k is integer and its minimum admissible value is $k = \lceil \log_2 n \rceil$. We shall prove the existence of the minimal solution by finding the set I. Let I' be the set $\{2^i \mid 0 \leq i \leq k-2\}$ and let $I = I' \cup \{n - 2^{k-1}\}$. The set I has k elements. For all $I_j \in \mathcal{P}(I)$, we have $0 \leq S(j) \leq n-1$. If $I_j = I$, the sum $S(j)$ takes its maximum value, namely $\sum_{i=0}^{k-2} 2^i + n - 2^{k-1} = 2^{k-1} - 1 + n - 2^{k-1} = n - 1$. This satisfies condition 2. Let us consider any integer p, $0 \leq p < 2^{k-1}$. Obviously, there is a subset I_j of I' such that $p = S(j)$. Let us consider $2^{k-1} \leq p < n$ and let $q = n - p - 1$. We have $0 \leq q < 2^{k-1}$ and, based on the same consideration as before, there is an $I_j \subset I'$ so that $q = S(j)$. In this case the sum of the elements of the set-difference $I \backslash I_j$ is exactly p. Consequently, $S(j)$ takes all the integer values in the interval $[0, n-1]$ and condition 3 is met. The set $I = \{1, 2, \ldots, 2^{k-2}, n - 2^{k-1}\}$ is a solution for the general problem of arbitrary n.

Our proof shows that, for each n, the running max filter can be computed by using the

Table 1: Solutions when n ranges in $[5, 16]$

n	Distinct solutions	Total
5	$\{1,1,2\}$	3
6	$\{1,1,3\}, \{1,2,2\}$	6
7	$\{1,2,3\}$	6
8	$\{1,2,4\}$	6
9	$\{1,1,2,4\}, \{1,1,3,3\}, \{1,2,2,3\}$	30
10	$\{1,1,2,5\}, \{1,1,3,4\}, \{1,2,2,4\}, \{1,2,3,3\}$	48
11	$\{1,1,3,5\}, \{1,2,2,5\}, \{1,2,3,4\}$	48
12	$\{1,1,3,6\}, \{1,2,2,6\}, \{1,2,3,5\}, \{1,2,4,4\}$	60
13	$\{1,2,3,6\}, \{1,2,4,5\}$	48
14	$\{1,2,3,7\}, \{1,2,4,6\}$	48
15	$\{1,2,4,7\}$	24
16	$\{1,2,4,8\}$	24

structure presented in Fig. 1. The computational complexity of the filter decreases from $O(n)$ to $O(\lceil \log_2 n \rceil)$. The number of intermediate operands that must be stored for each result is given by all the delays involved in the computation, i.e. $\sum_{i=1}^{k} m_i = n - 1$. The proof is constructive. It gives a particular solution for the delay operators $D^{m_1}, D^{m_2}, \ldots, D^{m_k}$, namely $D^1, D^2, \ldots, D^{2^{k-2}}, D^{n-2^{k-1}}$.

Any permutation of the set I of delay constants is also a solution of the problem. The entire set of solutions for any given n can be determined if we approach the problem as an integer programming one. This means to determine all the sets $\{m_1, m_2, \ldots, m_k\}$, $k = \lceil \log_2 n \rceil$, such that conditions 2 and 3 are fulfilled. The set of equations is:

$$S(j) = \sum_{m_j \in I_j} m_j = p \qquad (7)$$

where p takes all the values in $[0, n-1]$. Each m_i is bounded, $0 < m_i < n$. The problem may be solved by exhaustive search. The equations (7) lead to an efficient branch and bound of the solution space. The analysis of the complexity of the integer programming problem is beyond the purpose of our paper. However, for a broader view, we give in Table 1 all the distinct solutions when n ranges $[5, 16]$. The last column of Table 1 shows the total number of solutions derived by permutation.

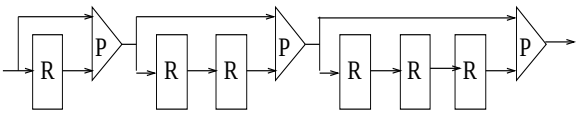

Figure 2: Pipeline implementation ($n = 7$).

3 Implementation

The flow graph presented in the previous section may be implemented either by software or hardware. For a hardware implementation, it is easy to see that the structure presented in Fig. 1 is merely a pipeline processor. Thus, branches become banks of registers and nodes become processing elements for maximum (minimum) computation. For a window size n, the pipeline has $k = \log_2 n$ stages (Fig. 2). Each stage consists of a processing element (P) and a bank of registers. The number of registers in each stage is given by the corresponding delay operator. The total number of registers is $n - 1$. A processing element may be regarded as a multiplexer driven by a comparator. The comparator selects the output of the multiplexer according to the result of the comparison between the two input operands. Since minimum may be approached as the maximum of the complemented sequence, the same structure can be used for running minimum filtering by complementing the data sequence both at the input and the output of the pipeline. The pipelining clock is determined by the processing time required by the P processors. At each clock tick an input sample is loaded and an output sample is delivered. The delay between the first sample loaded in the pipeline and its corresponding output sample equals n clock ticks. The pipeline registers are set to zero before loading the first sample.

A hard-wired dedicated processor for max/min filtering within a fixed window size is of limited interest in the context of signal and image processing applications where the size of the window filter has to take values in quite a large range. This requires flexible (programmable) processors for hardware implementations. Since, in almost all the applications the window size is fixed for a certain task, static reconfiguration of the pipeline can be used.

We have shown that for $2^{k-1} < n \leq 2^k$ the

Table 2: Delay constants and number of active stages for variable size window implementations. n ranges in $[3, 16]$

n	Delay constants	Active stages
3	$\{1, 1, 0, 0\}$	2
4	$\{1, 2, 0, 0\}$	2
5	$\{1, 2, 1, 0\}$	3
6	$\{1, 2, 2, 0\}$	3
7	$\{1, 2, 3, 0\}$	3
8	$\{1, 2, 4, 0\}$	3
9	$\{1, 2, 4, 1\}$	4
10	$\{1, 2, 4, 2\}$	4
11	$\{1, 2, 4, 3\}$	4
12	$\{1, 2, 4, 4\}$	4
13	$\{1, 2, 4, 5\}$	4
14	$\{1, 2, 4, 6\}$	4
15	$\{1, 2, 4, 7\}$	4
16	$\{1, 2, 4, 8\}$	4

pipeline processor has k stages. Each stage is composed by a P processor fed on with operands through two delay lines: an identity delay line and a fixed value delay line, m_i. The delay constants are $1, 2, \ldots, 2^{k-2}$ and $n - 2^{k-1}$. If we keep the first $k-1$ stages fixed and we modify the value of the delay of the last stage in the range $1 \leq m_k \leq n - 2^{k-1}$, we cover all the window sizes in the range $[2^{k-1}+1, n]$. Furthermore, if $m_k = 0$, both inputs to the P processor have the same value and, consequently, it has no effect in the computation. The structure operates with only $k-1$ stages. It follows, that a fully programmable max/min filter for any window size in $[2, 2^k]$ is implemented by using a programmable delay line for each stage. For stage i, $2 \geq i \geq k$, the delay line operates in the range $[0, 2^{i-1}]$. If the required window size w is in the range $2^{j-1} < w \leq 2^j$, the delay values are: $m_i = 0$ for $j + 1 \leq i \leq k$, $m_i = 2^{j-1} - (2^j - w) = w - 2^{j-1}$ for $i = j$ and $m_i = 2^{i-1}$ for $1 \leq i < j$, respectively. The configuration of delay registers for n ranges $[3, 16]$ is shown in Table 2.

For a pipeline implementation, the delay time introduced by propagating the signal through the stages with delay constants 0, has to be taken into account. Notice that in the worst case there are $k-1$ stages with zero delay constants. The throughput of the pipeline is a critical design parameter. If the extra delay time is not admissible, a solution still exists by multiplexing the output of each stage and by selecting only the output of the last working stage.

4 Conclusions

The cornerstone of our approach is given by the algebraic properties of maximum (minimum) operation, namely associativity, commutativity and idempotence. Our approach has produced a general computational structure (Fig. 1) that has been constrained to compute the desired filter output by specifying its elements, i.e. the number of processors and the values of delay constants. We have proven that for each window size, n, a set of delay constants exists, i.e. $\{1, 2, \ldots, 2^{k-2}, n - 2^{k-1}\}$, so that the computation is done in $k = \lceil \log_2 n \rceil$ comparisons per sample. The proposed computational structure leads to a pipeline implementation. A reconfigurable pipeline for variable size windows has been discussed. Besides maximum and minimum, other operators that have similar algebraic properties can be used on the same architecture (e.g. the greatest common divisor, the least common multiple, union and intersection).

References

[1] R. M. Haralick, S. R. Sternberg, X. Zhuang, "Image Analysis Using Mathematical Morphology", *IEEE Trans. on PAMI*, vol. PAMI-9, no. 4, pp. 532-530, July 1987.

[2] M. Werman, S. Peleg, "Min-Max Filters in Texture Analysis", *IEEE Trans. on PAMI*, vol. PAMI-7, pp. 730-733, 1986.

[3] P. W. Verbeek, H. A. Vrooman, L.J. Vliet, "Low-Level Image Processing by Max-Min Filters", *Signal Processing*, vol. 15, no. 3, pp. 249-258, Oct. 1988.

[4] I. Pitas, "Fast Algorithms for Running Ordering and Max/Min Calculation", *IEEE Trans. on Circuits and Systems*, vol. 36, no. 6, pp. 795-804, 1989.