SYNTHESIS OF A PARALLEL, OPTIMAL STACK FILTER TRAINING ALGORITHM

Kelvin L. Fong, George B. Adams III, Edward J. Coyle

School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907
{kelvin, gba, coyle}@purdue.edu

Jisang Yoo

Electronics Dept., Hallym University
1, Okcheon-Dong
Chuncheon, KOREA
jsyoo@sun.hallym.ac.kr

ABSTRACT

An adaptive algorithm for generating optimal stack filters is presented. The algorithm is iterative and highly parallel. The algorithm is summarized, its time complexities are analyzed, and implementation details, such as data distribution and communication patterns, are described including performance results from an implementation on a 16K processor MasPar MP-1 SIMD computer.

1. INTRODUCTION

Stack filters are a class of sliding window, nonlinear filters. One of the main strengths of stack filters is the existence of an analytical technique for determining a stack filter which is optimal for estimation under the mean absolute error criterion [4, 5]. Although these results provide a systematic approach for designing an optimal stack filter, knowledge of the joint threshold crossing statistics of the signal and noise processes is required. Such knowledge is rarely available in practice, particularly in image processing applications.

To alleviate these problems, adaptive stack filter training algorithms were developed. With these algorithms, an optimal stack filter can be determined via observations of training sequences. Their principal limitations are their excessive computational complexity and lack of significant parallelism. These factors prevent both quick generation of stack filters and any significant speedup from execution on highly parallel computers.

Stack filters are defined by two properties: the weak superposition property known as the *threshold decomposition property*, and an ordering property known as the *stacking property* [7].

This work was supported by NSF Grants CDA-9015696, CDA-9422250, and CDA-9617388.

Threshold decomposition allows analysis of a digital filter to be broken down into its binary threshold components. Let x_l be defined as the binary image obtained by thresholding the image X at level l. Let $T_l(\cdot)$ denote this thresholding operator, so that $x_l = T_l(X)$. Each binary pixel of the image is denoted $x_l(i,j)$, where i,j are the coordinates of the binary pixel on the lth threshold level. Therefore $X(i,j)$ and $x_l(i,j)$ are related such that

$$x_l(i,j) = \begin{cases} 1, & \text{if } X(i,j) \geq l \\ 0, & \text{otherwise.} \end{cases}$$

In this manner, X is the sum of its individual thresholded component levels:

$$X = \sum_{l=1}^{M} x_l = \sum_{l=1}^{M} T_l(X),$$

where M is the number of quantization levels of X. A windowed section, W, of the image may be similarly defined, and possesses the same decomposition properties:

$$W[i,j] = \sum_{l=1}^{M} T_l(W[i,j]) = \sum_{l=1}^{M} w_l[i,j]$$

where $w_l[i,j] = T_l(W[i,j])$ is the binary array obtained by thresholding the pixels in window $W[i,j]$, referenced at the coordinates i,j, at level l.

The stacking property imposes an ordering condition onto the filter's output. In general, the stacking property requires that for any stack filter binary output equal to 1 at threshold level l, all outputs on the levels below (less than) l must also be 1 for that window position. The stacking property is defined as a partial ordering of arrays, where $w_l[i,j]$ stacks on top of $w_{l-1}[i,j]$ if every element $w_l[i,j]$ is less than or equal to the corresponding element in $w_{l-1}[i,j]$. In other words,

$$w_l[i,j] \leq w_{l-1}[i,j], \quad l = 1, 2, \ldots M.$$

Note that this is a partial ordering since not all binary patterns are related in this manner.

A common measure for stack filter evaluation is the mean absolute error (MAE) criterion. The MAE criterion is determined by comparing the output $\hat{X}(i,j)$, which is the filter's estimate of $X(i,j)$, based on the windowed observation $W[i,j]$ so that

$$\text{MAE} = E\left[\left|X(i,j) - \hat{X}(i,j)\right|\right].$$

The filter generating the $\hat{X}(i,j)$ that minimizes the MAE is the optimal stack filter [4, 5].

2. ALGORITHM

Our training algorithm and its subroutines are presented in Figures 1 and 2. The training algorithm TRAIN takes as input parameters the original image, X^{orig}, the corrupted image, X^{corr}, the window size and shape, n, and the images' size, x and y, and depth, z. The bulk of the computation occurs in the three subroutines DETERMINE-D^{opt}, STACK, and THRESHOLD.

TRAIN$(X^{orig}, X^{corr}, n, x, y, z)$
1 $D^{opt} \leftarrow$ DETERMINE-$D^{opt}(X^{orig}, X^{corr}, n, x, y, z)$
2 $k \leftarrow 0$
3 $D^{(k)} \leftarrow \vec{0}$
4 $\tilde{D}^{(k)} \leftarrow D^{(k)} + D^{opt}$
5 $D^{(k+1)} \leftarrow$ STACK$(\tilde{D}^{(k)}, n)$
6 **if** $D^{(k+1)}$ has not converged
7 **then** $inc(k)$
8 **goto** Step 4
9 $S \leftarrow$ THRESHOLD$(D^{(k+1)}, n)$
10 **return** S

Figure 1: Stack filter training algorithm.

2.1. Analyses

TRAIN begins by determining the optimal decision vector, D^{opt}, and initializing the constrained decision vector (a decision vector constrained to obey the stacking property), $D^{(k)}$. TRAIN then iterates on $D^{(k)}$ until it has adequately converged. Step 4 updates $D^{(k)}$ by adding D^{opt} to create the unconstrained decision vector $\tilde{D}^{(k)}$. Step 5 enforces the global stacking property on $\tilde{D}^{(k)}$ to produce $D^{(k+1)}$. If the MAE of $D^{(k+1)}$ has suitably converged then $D^{(k+1)}$ is thresholded and returned; otherwise another iteration begins. A proof that TRAIN converges to the optimal stack filter is given in [8].

DETERMINE-$D^{opt}(X^{orig}, X^{corr}, n, x, y, z)$
1 $D^{opt} \leftarrow \vec{0}$
2 **for** $i \leftarrow 1$ **to** x
3 **do for** $j \leftarrow 1$ **to** y
4 **do for** each threshold level l
5 **do if** $w_l^{orig}(i,j) \geq w_l^{corr}(i,j)$
6 **then** $inc(d^{opt}_{w_l^{corr}(i,j)})$
7 **else** $dec(d^{opt}_{w_l^{corr}(i,j)})$
8 **return** D^{opt}

STACK(D, n)
1 $i = 1$
2 **while** $i \leq 2^{n-1}$
3 **do for** $j \leftarrow 0$ **to** $2^n - 1$ **by** $2i$
4 **do for** $k \leftarrow 0$ **to** $i - 1$
5 **do if** $d_{k+j} > d_{i+j+k}$
6 **then** $d_{j+k} \leftarrow \lfloor \frac{d_{j+k} + d_{i+j+k}}{2} \rfloor$
7 $d_{i+j+k} \leftarrow \lceil \frac{d_{j+k} + d_{i+j+k}}{2} \rceil$
8 $i \leftarrow 2i$
9 **return** D

THRESHOLD(D, n)
1 **for** $i \leftarrow 0$ **to** $2^n - 1$
2 **do if** $d_i \geq 0$
3 **then** $d_i \leftarrow 1$
4 **else** $d_i \leftarrow 0$
5 **return** D

Figure 2: TRAIN algorithm subroutines.

The algorithm's operation is illustrated geometrically in Figure 3. Steps 4-8 in TRAIN iteratively updates the current decision vector and then enforces the stacking property. This may be viewed as a series of repeated approximations to force the direction of the current constrained decision vector to converge towards the direction of the optimal decision vector. Step 4 moves $D^{(k)}$ towards D^{opt}, but may create stacking violations in $\tilde{D}^{(k)}$. Step 5 must resolve these violations while leaving $D^{(k+1)}$ closer to D^{opt} than $D^{(k)}$.

2.2. Data Representation and Distribution

The PRAM and MP-RAM parallel models [1] are used in this paper's time complexity analysis. Two MP-RAM interconnections are assumed; an all-to-all (crossbar) connected network, and a NEWS mesh connected network. These are illustrated in Figure 4.

The original and corrupted images are stored as binary encoded data. Each decision variable is repre-

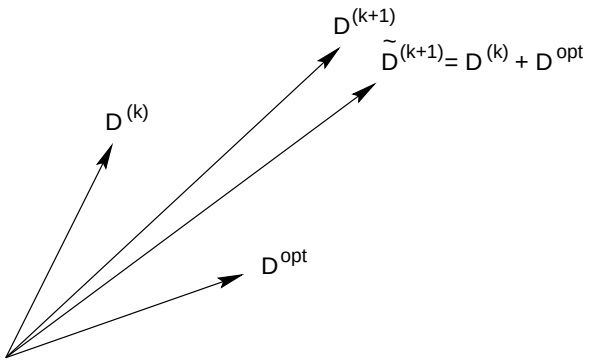

Figure 3: Training algorithm operation.

sented as an integer; hence, the decision vectors, D^{opt} and D^k, each require 2^n integers. Hence, for the MP-RAM models, images and decision vectors may either be partitioned across the processors or duplicated in each processor's local memory. In practice sufficient memory may not exist to store the entire decision vector in local memory. The choice of data distribution on the MP-RAM may affect the performance of the algorithm, as described later in section 3.2.

2.3. Time Complexity

This section analyses the efficiency of TRAIN by measuring the time rate of growth to train a stack filter for a given image pair and window size. The notation $\Theta(n)$ will be used with the following definition. $\Theta(g(n)) = f(n)$: There exist constants c_1, c_2 such that for all n, $0 < c_1 g(n) < f(n) < c_2 g(n)$. See [3] for more information on asymptotic notation.

Steps 2 and 7 of TRAIN simply set and increment the iteration counter variable, and each have time complexity $\Theta(1)$ for both serial and parallel models.

DETERMINE-D^{opt} first initializes all 2^n decision variables, d_i^{opt}, in D^{opt} to 0; this requires $\Theta(2^n)$ operations. The operations are independent of each other, so the parallel time complexity is $\Theta(\frac{2^n}{p})$ for both the PRAM and MP-RAM models. The remainder of the subroutine compares each pixel's windowed threshold level, $w_l[i,j]$, in the corrupted image to the original. For a binary encoding, where each of the $x \times y$ pixels may take on any value from $0 \ldots 2^z - 1$, this requires $x \times y \times (2z\lg(z) + n^2)$ observations, and either an increment or decrement to the appropriate decision variable. The thresholding details are omitted here for brevity. Each observation and update is independent of the others, so the complexity of Steps 3-7 is $\Theta(xy(2z\lg(z) + n^2))$ on a RAM, and $\Theta(\frac{xy(2z\lg(z)+n^2)}{p})$ on both PRAM and MP-RAM models.[1] Assuming that $2^n >> xy(2z\lg(z) + n^2)$, then the time complexity of DETERMINE-D^{opt} is $\Theta(2^n)$ on a RAM, and $\Theta(\frac{2^n}{p})$ on the PRAM. The individual decision variables in each PE's local memory must be combined in MP-RAM memory. If each decision variable is combined across the PEs, then placed in its assigned PE's memory; this requires $\Theta(\lg(p))$ communications and computations for each decision variable. The time complexity of DETERMINE-D^{opt} on an MP-RAM is therefore $\Theta(\frac{2^n}{p} + 2^n\lg(p)) = \Theta(2^n\lg(p))$.

STACK operates on the unconstrained decision vector, $\tilde{D}^{(k)}$, so its complexity is a function of the 2^n de-

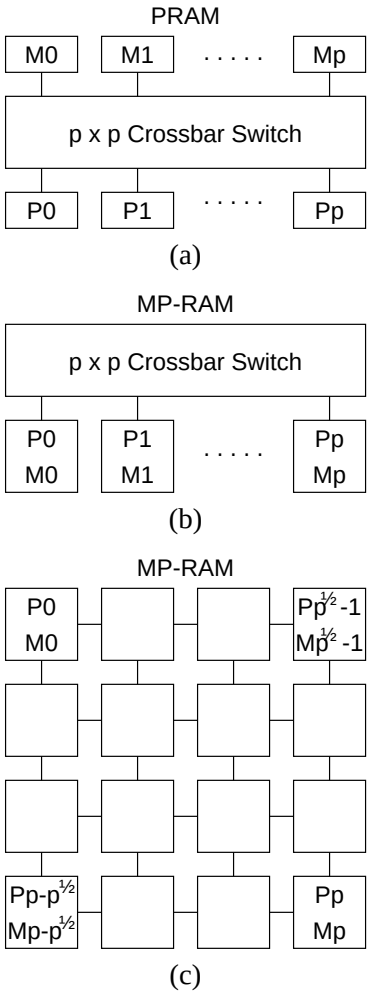

Figure 4: The three parallel computation models considered: (a) PRAM, (b) MP-RAM with an all-to-all interconnection, (c) MP-RAM with a mesh interconnection.

[1] The time for MP-RAM interprocessor communications to transfer a window's overlapping pixels between processors is omitted in this analysis.

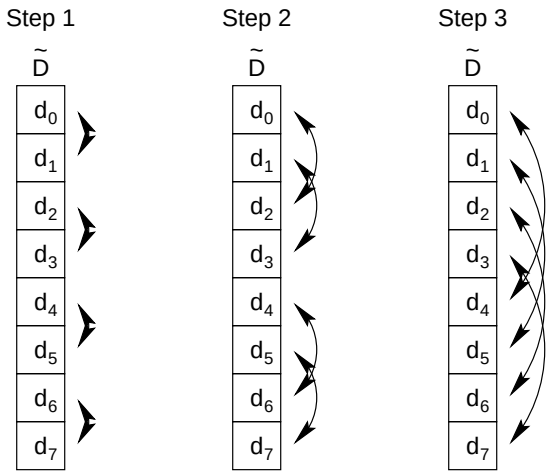

Figure 5: Three steps, each parallel, to enforce the stacking property for window size $n = 3$.

cision variables composing $\tilde{D}^{(k)}$. The bulk of the computation occurs in Steps 6-7, within the triply nested loop. Step 2 will iterate n times. The number of iterations performed by Steps 3-4 is a function of the iteration variable i. Consider then for each iteration of Step 2, each pair of decision variables in $\tilde{D}^{(k)}$ must be examined; this requires 2^{n-1} operations. Whether or not a stacking violation is found, each decision variable pair must be examined in Step 5. Hence the time complexity of STACK is $n2^{n-1}$ or $\Theta(n2^n)$ on a RAM. On a PRAM Steps 3-4 are fully parallelizable, so the PRAM complexity is $\Theta(\frac{n2^n}{p})$. Parallelizing the stacking corrections on an MP-RAM requires data movements between processor pairs sending and receiving decision variables to be compared. This requires $\lceil \frac{2^{n-1}}{p} \rceil \times (n-1)$ parallel message exchanges, so the MP-RAM complexity is $\Theta(\lceil \frac{2^{n-1}}{p} \rceil (n-1) + \frac{n2^n}{p})$ or $\Theta(\frac{n2^n}{p})$. Figure 5 illustrates the three parallel steps in STACK required to impose the stacking property onto a decision vector for a window size $n = 3$ filter.

THRESHOLD simply rescales the decision vector to a binary vector; the resulting binary vector is the final stack filter. Every decision variable is examined and set to 1 if greater than or equal to zero, or 0 if less than zero. No interprocessor communications are required, so the RAM, PRAM, and MP-RAM time complexities are $\Theta(2^n), \Theta(\frac{2^n}{p})$, and $\Theta(\frac{2^n}{p})$, respectively.

The time complexity of TRAIN is summarized in Table 1. As long as the number of iterations performed in TRAIN is much less than 2^n, which we have observed to be the case in practice, the asymptotic complexities are not affected by the number of iterations performed (though the execution time may certainly be).

Table 1: TRAIN time complexity.

Routine	Complexity		
	RAM	PRAM	MP-RAM
TRAIN	$\Theta(n2^n)$	$\Theta(\frac{n2^n}{p})$	$\Theta((\lg(p) + \frac{n}{p})2^n)$
DET-D^{opt}	$\Theta(2^n)$	$\Theta(\frac{2^n}{p})$	$\Theta(2^n \lg(p))$
STACK	$\Theta(n2^n)$	$\Theta(\frac{n2^n}{p})$	$\Theta(\frac{n2^n}{p})$
THRESHOLD	$\Theta(2^n)$	$\Theta(\frac{2^n}{p})$	$\Theta(\frac{2^n}{p})$

3. IMPLEMENTATION

3.1. The MasPar MP-1

The MasPar MP-1 [2] SIMD computer operated by Purdue University's Parallel Processing Laboratory is configured with 16,384 processor elements (PEs). The data parallel unit includes the array control unit, PE array, and communication mechanisms. Each PE has 16 Kbytes of RAM available. The PE array is a 2D matrix representation of all the PEs in the system. A system has 1K, 2K, 4K, 8K, or 16K PEs that are arranged in a matrix with either an equal number of columns and rows or one that has twice as many columns as rows. The PEs are arranged in clusters of nonoverlapping 4×4 matrices of 16 PEs and 16 processor memories per cluster.

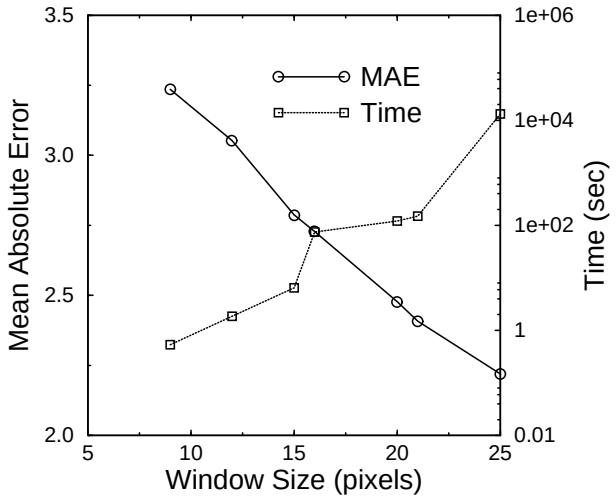

Figure 6: MAE and training time plotted as a function of windows size for *einstein* and *einsteinI10* images.

Figure 6 is a plot of the MAE for stack filters generated with TRAIN for window sizes ranging from 9 to 25, and their corresponding training times. MAE decreases somewhat linearly, while their corresponding

training times increase somewhat exponentially, with increasing window size.

3.2. Decision Variable Observations

Two issues, regarding the relative sizes of the images used to train the filter and the size of the filter window, affected both the implementation of the algorithm and the performance of the algorithm on the MasPar. For example, training a 5×5 pixel filter for a $512\times512\times8$ bit image yields a decision vector with $2^{25} = 33,554,432$ decision variables while $512 \times 512 \times 2^8 = 67,108,864$ windowed threshold levels will be observed (or fewer, when excluding image border pixels). Our experiments revealed many stacking violations in D^{opt} for this case. Furthermore for the $512 \times 512 \times 8$ bit *einstein* images training a 5×5 filter, only 1,762,593 unique decision variables were observed. On the other hand, training a 3×3 pixel filter on the same image yields a decision vector composed of $2^9 = 512$ decision variables. For the $256 \times 256 \times 8$ and $512 \times 512 \times 8$ size images we examined, D^{opt} possessed no stacking violations for the 3×3 window size.

Our experiments support the interpolation between these two data points. When the decision vector is underobserved (decision variables are not observed when determining D^{opt}) many stacking violations result in D^{opt}, while the converse (all decision variables are observed when determining D^{opt}) results in few if any stacking violations in D^{opt}. If D^{opt} possesses no stacking violations, D^{opt} corresponds to the optimal stack filter, and TRAIN converges after one iteration. If D^{opt} possesses many stacking violations, the optimal stack filter differs significantly from D^{opt} and many iterations are required to converge to the optimal constrained decision vector.

Also the most efficient manner of combining the individual PE observations into D^{opt} on an MP-RAM may vary depending on whether D^{opt} is underobserved. For example, if D^{opt} is fully observed, each decision variable may be combined and placed into its respective PE's memory. If D^{opt} is largely underobserved, this will result in few processors doing useful work in the combining process, because most decision variable values will be zero. In this case, having each processor with an observed decision variable send its value to the destination processor which then sums its value may be more efficient. This proved to be the case on the MasPar.

Finally if D^{opt} is largely underobserved, memory space may be saved by using a hash table to represent D^{opt}. This would reduce the data space requirements by almost 128 Mbytes when training a 25 pixel filter and D^{opt} is represented as an `int` array, while increasing access times to D^{opt} by a constant factor. This technique was not implemented in our experiments.

4. CONCLUSIONS

Because the operations on the training images and D^{opt} are largely independent, and because the size of the training images and D^{opt} are fairly large, a great deal of parallelism may be exploited in TRAIN. The bulk of the computation for large filter sizes occurs in the repeated calls to the STACK subroutine. The MasPar MP-1 computer proved to be an acceptable platform for implementing TRAIN, with all processors enabled for much of the algorithm when training filters of size 15 and greater. Finally the relative sizes of the training image and the filter window significantly affects both the performance and implementation of the algorithm, in terms of the number of iterations required to converge to the optimal stack filter, the method of combining D^{opt} on an MP-RAM multiprocessor, and the data representation of D^{opt}.

5. REFERENCES

[1] George S. Almasi and Allan Gottlieb. *Highly Parallel Computing.* The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA 94065, 1994.

[2] T. Blank. "The MasPar MP-1 Architecture" in *Compcon Spring 1990*, pp. 20-24, February 1990.

[3] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* The MIT Press, Cambridge, MA, 1991.

[4] E. J. Coyle and J.-H. Lin. "Stack Filters and the Mean Absolute Error Criterion" in *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 36, pp. 1244-1254, August 1988.

[5] E. J. Coyle, J.-H. Lin, and M. Gabbouj. "Optimal Stack Filtering and the Estimation and Structural Approaches to Image Processing" in *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 37, pp. 2037-2064, December 1989.

[6] J.-H. Lin, T. M. Selke, and E. J. Coyle. "Adaptive Stack Filtering Under the Mean Absolute Error Criterion" in *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 38, pp. 938-954, June 1990.

[7] P. D. Wendt, E. J. Coyle, and N. C. Gallagher, Jr. "Stack Filters" in *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 34, pp. 898-911, August 1986.

[8] J. Yoo. "Stack Filters: Design, Algorithms, and Applications." Ph.D. Dissertation, Purdue University, August 1993.