EFFICIENT SERIAL AND PARALLEL ALGORITHMS FOR MEDIAN FILTERING [+]

Sanjay Ranka [*] *and Sartaj Sahni*

University of Minnesota

Abstract

We develop a serial algorithm for separable median filtering that requires only two comparisons per element when the window size is three. In addition, fast parallel CREW PRAM algorithms with good processor-time product are developed for separable median filtering and two dimensional median filtering.

Keywords and Phrases

Median filtering, separable median filtering, complexity, CREW PRAM algorithms

1 INTRODUCTION

Median filters are widely used for smoothing operations in signal, speech, and image processing ([TUKE76], [JAYA76]). This filtering operation is performed on an $N \times N$ image matrix $I[1..N, 1..N]$ using a $W \times W$ window where $W = 2w + 1$ is an odd number. The result of median filtering is an $N \times N$ matrix MEDIAN2D defined as:

$$MEDIAN2D[i, j] = median\{I[a, b] \mid nbhd(a, i, w, N) \text{ and } nbhd(b, j, w, N)\}$$

where

$$nbhd(p, q, r, s) = \begin{cases} true & (p - q) \bmod s \le r \text{ or } (q - p) \bmod s \le r \\ false & otherwise \end{cases}$$

A straightforward serial computation of MEDIAN2D takes $O(N^2W^2)$ time. This is easily reduced to $O(N^2 W \log W)$ by using balanced search trees. Huang, Yang, and Tang [HUAN79] have developed an $O(N^2W)$ algorithm for the case when the image values are in the range 0 through $K-1$ (i.e., there are K gray levels). Narendra [NARE81] has introduced a related filtering operation, *separable median filter*, that can be computed in $O(N^2 \log W)$ serial time (The algorithm presented in [NARE81] for this takes $O(N^2W)$ time. This was improved to $O(N^2 \log W)$ by Basu and Brown [BASU87]). The separable median filter, SMEDIAN2D, is obtained by computing two one dimensional medians as below:

$$MEDIAN1D[i, j] = median\{I[a, j] \mid nbhd(a, i, w, N)\}$$

$$SMEDIAN2D[i, j] = median\{MEDIAN1D[i, b] \mid nbhd(b, j, w, N)\}$$

In this formulation, row medians are computed first and then column medians. An alternate is to compute column medians first and then row medians. Changing the order in this way will generally lead to different results.

Basu and Brown [BASU87] also develop special purpose hardware for separable median filtering. Parallel algorithms to compute MEDIAN2D on a pyramid computer have been proposed in [TANI82] and [STOU83]. Note that by using the $O(\log\log M)$ time M processor algorithm of

+ This research was supported in part by the National Science Foundation under grants DCR84-20935 and MIP 86-17374
* Professor Ranka's current address is CIS Dept, 313 Link Hall, Syracuse University, Syracuse, NY 13244.

[AJTA86] to find the median of m elements, we can calculate SMEDIAN2D and MEDIAN2D in $O(\text{loglog}W)$ time using $O(N^2W)$ processors and $O(N^2W^2)$ processors respectively. The resulting processor-time products are $O(N^2W\text{loglog}W)$ and $O(N^2W^2\text{loglog}W)$. We develop algorithms which have good processor-time product (i.e., within polylogarithmic factor of the optimal serial algorithm). The results obtained in this paper are summarized below:

(1) In Section 2, we present an algorithm to compute MEDIAN1D for the case $W = 3$. This algorithm requires at most two comparisons to compute each element of MEDIAN1D. For the case $W = 3$, Basu and Brown [BASU87] present an algorithm that requires at most 2.5 comparisons per element of MEDIAN1D.

(2) In Section 3, we show that $O(\log W)$ comparisons per element of MEDIAN1D is a lower bound under the decision tree model. Hence, the algorithm of [BASU87] is asymptotically optimal for this problem.

(3) Parallel CREW PRAM algorithms for the separable median filter problem are developed in Section 4. One of these computes MEDIAN1D and hence SMEDIAN2D in $O(\log^2 W\text{loglog}W)$ time using $O(N^2/(\log W\text{loglog}W))$ processors. The processor time product for this algorithm is $O(N^2\log W)$ which by the result of Section 3 is optimal. The other algorithm developed in this section computes MEDIAN1D in $O(\log^2 W)$ time using $O(N^2\log W)$ processors. The processor-time product for this algorithm is $O(N^2\log^3 W)$ which is suboptimal.

(4) The technique used in Section 4 is extended in Section 5 to obtain an $O(\log^2 W)$ CREW PRAM algorithm to compute MEDIAN2D. This algorithm uses $O(N^2\log^2 W)$ processors. Notice that the processor-time product of this algorithm is $O(N^2\log^4 W)$ compared to $O(N^2W\log W)$ for the serial algorithm using balanced search trees. We can obtain a serial $O(N^2\log^4 W)$ algorithm for MEDIAN2D by simulating our parallel CREW PRAM algorithm.

2 COMPUTING MEDIAN1D SERIALLY

Basu and Brown [BASU87] show that when $W = 3$ MEDIAN1D can be computed using at most 2.5 comparisons per element of MEDIAN1D. Their algorithm is presented in Figure 1. For simplicity, this C code computes only $median1d[1..N-2]$ for the N pixel image $image[0..N-1]$. In iteration i of the for loop, two $median1d$ values ($median1d[i+1]$ and $median1d[i+2]$) are computed. First, in (a, b) we save $image[i+1]$ and $image[i+2]$ in ascending order. Now, $median1d[i+1]$ is the *median* of $\{image[i], a, b\}$ and $median1d[i+2]$ is the *median* of $\{image[i+2], a, b\}$. Since a, b are in ascending order, each median is computed using the respective if statement of Figure 1.

```
for (i = 0; i < N-3 ; i += 2){
    if (image[i+1] < image[i+2]) {
       a = image[i+1]; b = image[i+2]; }
    else { a = image[i+2];
         b = image[i+1] ;}
    if (image[i] < a) median1d[i+1] = a;
      else if (image[i] > b)  median1d[i+1] = b;
           else median1d[i+1] = image[i];
    if (image[i+3] < a) median1d[i+2] = a;
      else if (image[i+3] > b)  median1d[i+2] = b;
           else median1d[i+2] = image[i+3];
}
```

Figure 1: Basu and Brown's algorithm ($W = 3$)

```
if (image[0] < image[1])  max = 1 ;
  else max = 0;
for (i = 2; i < N-1 ; i++){
    min = 2 * i - 3 - max;
    if (image[i] < image[min] ) {
      max = i-1; median1d[max] = image[min];  }
    else if (image[i] > image[max] ) {
          median1d[i-1] = image[max];
          max = i; }
        else { median1d[i-1] = image[i];
          max = min + 1; }
}
```

Figure 2: Finding the median ($W = 3$)

In each iteration of the for loop of Figure 1, two $median1d$ values are computed and at most

5 comparisons are performed. Hence, at most 2.5 comparisons are performed per element of *median* 1*d*. On the average, however, each iteration requires only 13/3 comparisons (the probability of each *image* [ ] > *b* compare being 2/3) and the comparisons per element of *median* 1*d* becomes 13/6. In Figure 2, we present an alternate way to compute *median* 1*d*. Once again the code computes only *median* 1*d* [1..*N* −2]. *median* 1*d* [*i* − 1] = *median*{*image* [*i* − 2], *image* [*i* − 1], *image* [*i*]}, $1 \le i \le N-2$ At the start of each iteration of the for loop of Figure 2, *max* is such that *image* [*max*] is the larger of *image* [*i* − 2] and *image* [*i* − 1]. The smaller of these two is given by the formula 2*i* − 3 − max. The if statement in the for loop uses this information to compute *median* 1*d* [*i* − 1] and also to update max for the next iteration. The maximum number of comparisons per iteration of the for loop is 2 and the average (assuming a 2/3 probability for the *image* [ ] > *image* [*max*] comparisons) is 5/3.

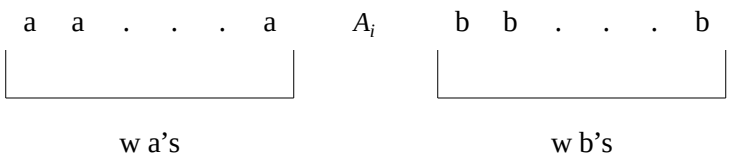
The code of Figures 1 and 2 was run on an Apollo DN330 workstation. For worst case data (*image* [0..*N* −1] = (2 1 2 2 4 3 5 4 6 ...) for Figure 1) and (*image* [0..*N* −1] = (1 *N* 2 *N* −1 3 *N* −2 4 *N* −3 ....)) for Figure 2) and *N* = 16024, Figure 1 took 0.4 seconds while Figure 2 took 0.376 seconds. Hence our new algorithm takes 6% less time than the algorithm of Figure 1 on worst case data. The average run time of Figure 2 is about 2.4% less than that of Figure 1. While our new algorithm runs marginally faster than the algorithm of [BASU87], its main contribution is theoretical. It uses fewer comparisons.

3 A LOWER BOUND FOR MEDIAN1D

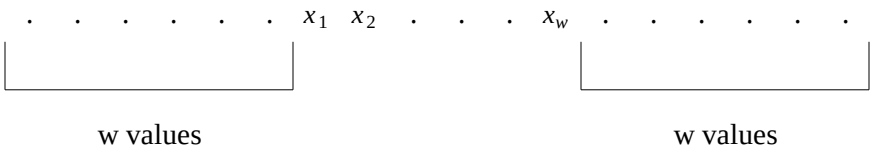
Theorem 1: For the decision tree model of computation ([HORO86], pp. 341), computing MEDIAN1D requires Ω(log*W*) comparison, on average, per element in MEDIAN1D.

Proof: The proof is similar to that used by Shamos [SHAM76] to show that Ω(*nlogn*) is a lower bound for the online medians problem.

Let $A_1 \cdots A_{3w}$ be 3w independent sequences of *w* distinct numbers each. Let *a* and *b* be such that *a* is smaller than every number in the A_i's and *b* is greater than every number in the A_i's. Construct the *N*×*N*, *N* = 3*w*, image matrix I such that row *i* is:



One may verify that the i'th row of MEDIAN1D with $W = 2w + 1$ has the form:



where $x_1 \cdots x_w$ are the numbers of A_i in ascending order. So, using this construction, we can use every algorithm for MEDIAN1D to sort $3w$ sequences of size w. The number of different possible outcomes for such a sort is $(w!)^{3w}$. Under the decision tree model [HORO86], every algorithm to sort these $3w$ sequences must have average depth $\Omega(3w^2 \log w)$. Consequently, under this model, every algorithm to compute MEDIAN1D must have average complexity $\Omega(3w^2 \log w)$. So, $\Omega(\log w) = \Omega(\log W)$ work must be done, on average, per element of MEDIAN1D. This result can be easily extended to the case $N = 6w, 9w, \ldots$ □

4 CREW PRAM ALGORITHMS FOR SEPARABLE MEDIAN FILTER

Suppose we have an $N \times N$ image I and a $1 \times W$ window, $W = 2w + 1$. The fastest way to compute MEDIAN1D on a CREW PRAM is to use W processors per element of MEDIAN1D. Each group of W processors finds $MEDIAN1D[i, j]$ for a distinct index pair (i, j). This is done in $O(\log\log W)$ time by using the algorithm of [AJTA86] to compute the median of the elements

$$\{I[a, j] \mid nbhd(a, i, w, N)\}$$

The processor-time product for this algorithm is $O(N^2 W \log\log W)$ which is not optimal. The processor-time product for the asymptotically optimal algorithm of [BASU87] is $O(N^2 \log W)$. In this section, we develop two parallel algorithms for MEDIAN1D (and hence SMEDIAN2D) with processor time-products $O(N^2 \log^3 W)$ and $O(N^2 \log W)$ respectively. The first has a run time $O(\log^2 W)$ and uses $O(N^2 \log W)$ processors. The run time of the second algorithm is

$O(\log^2 W \log\log W)$ and the processor requirement is $O(N^2/(\log W \log\log W))$. So, while the second algorithm has an optimal processor-time product, its run time is not optimal. Neither the run time nor the processor-time product of our first algorithm is optimal. However, the first algorithm is asymptotically faster then the second and serves as an introduction to the second. The existence of an $O(\log\log W)$ CREW PRAM algorithm with optimal processor-time product remains an open problem.

4.1 First Algorithm

In this algorithm, $MEDIAN1D[i, j]$ is computed independently for each row i. Let $x_1, x_2, \cdots, x_N$ be the image values in some row i of I. For simplicity, we assume that w divides N and $N/w > 2$. Partition $x_1, ..., x_N$ into N/w segments $S_1, S_2, ..., S_{N/w}$. This is done left to right with $x_1, ..., x_w$ in S_1; $x_{w+1}, ..., x_{2w}$ in S_2; etc. The steps in Algorithm 1 are:

Step1: Let $Y_a = S_{(a-2) \bmod N/w + 1} \cup S_a \cup S_{a \bmod N/w + 1}$, $1 \le a \le N/w$. Sort each Y_a.

Step2: Let $rank(left, j)$ = position of x_j in $Y_{(a-2) \bmod N/w + 1}$,

$rank(middle, j)$ = position of x_j in Y_a,

$rank(right, j)$ = position of x_j in $Y_{(a) \bmod N/w + 1}$,

where $a = \lceil j/w \rceil$

Step3: Sort each S_b by repeatedly merging adjacent subsequences of size at most 2^j, $0 \le j < \log_2 w$. The merged subsequences for each j are saved.

Step4: For each l, $1 \le l \le N$ form the subsequence set, Z_l, (from the results of Step3) that consists of the $W = 2w + 1$ elements x_c, $nbhd(c, l, w, N)$.

Step5: Search Z_l for the median element. This element is $MEDIAN1D[i, l]$.

Consider the case $w = 4$, and $N = 16$. Let $[x_1, ..., x_{16}]$ be [2, 4, 6, 1, 3, 9, 8, 7, 11, 14, 13, 5, 10, 12, 16, 15]. For this, we get:

Y_1 = 10, 12, 16, 15, 2, 4, 6, 1, 3, 9, 8, 7

Y_2 = 2, 4, 6, 1, 3, 9, 8, 7, 11, 14, 13, 5

Y_3 = 3, 9, 8, 7, 11, 14, 13, 5, 10, 12, 16, 15

Y_4 = 11, 14, 13, 5, 10, 12, 16, 15, 2, 4, 6, 1

Let SY_i represent the sorted version of Y_i following Step 1. We have:

SY_1 = 1, 2, 3, 4, 6, 7, 8, 9, 10, 12, 15, 16

SY_2 = 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 13, 14

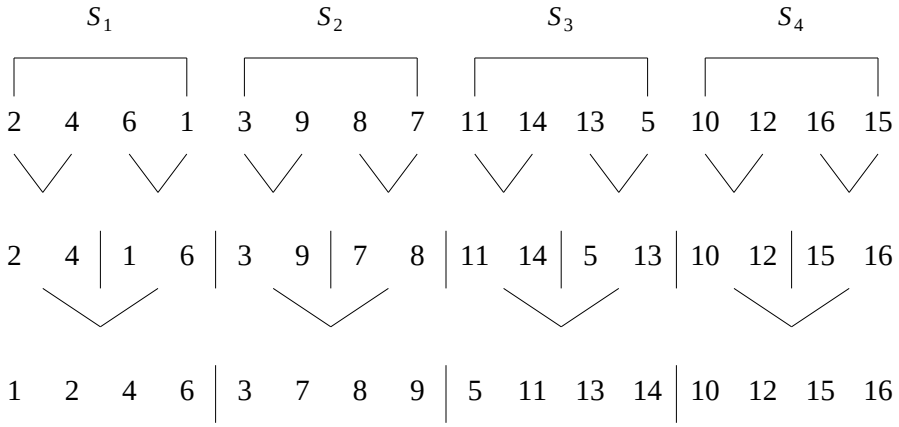
SY_3 = 3, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16

SY_4 = 1, 2, 4, 5, 6, 10, 11, 12, 13, 14, 15, 16

The quadruples (X_i, $rank(left, i)$, $rank(middle, i)$, $rank(right, i)$) for $1 \le i \le 16$ are:

(1, 1, 1, 1)	(5, 5, 2, 4)	(9, 8, 9, 5)	(13, 11, 9, 9)
(2, 2, 2, 2)	(6, 5, 5, 6)	(10, 6, 6, 9)	(14, 12, 10, 10)
(3, 3, 3, 1)	(7, 6, 7, 3)	(11, 10, 7, 7)	(15, 11, 11, 11)
(4, 3, 4, 4)	(8, 7, 8, 4)	(12, 8, 8, 10)	(16, 12, 12, 12)

The step 3 pairwise merging takes the form:



For step 4, consider the case $l = 6$. The elements x_l are $\{x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}\}$. The necessary sorted subsequences are:

$$Z_6 = \begin{cases} 4 \\ 1\ 6 \\ 3\ 7\ 8\ 9 \\ 11\ 14 \end{cases}$$

It is easy to see that $|Z_l|$ is $O(\log w)$.

To find the median of these $W = 2w + 1$ elements we replace each x_i in Z_l by a pair (x_i, $rank_i$)

```
low =1 ; high = 3w; found = false;
while not found do
begin
        mid = ⌊(low + high)/2⌋ ;
        perform a binary search in each of the sequences
        of Z_l to determine the number of elements in
        each sequence with rank_i < mid;
        add all these counts;
        if this sum equals w
        then [element with rank mid is the median; found = true]
        else if sum < w then set low = mid
                        else set high = mid;
end;
```

Figure 3: Finding the median in Z_l

where $rank_i$ is as given below:

$$rank_i = \begin{cases} rank(left, i) \text{ if } x_i \in S_{(a-2) \bmod N/w + 1} \\ rank(middle, i) \text{ if } x_i \in S_a \\ rank(right, i) \text{ if } x_i \in S_{a \bmod N/w + 1} \end{cases}$$

where $a = \lceil l/w \rceil$. For our example, the transformed Z_6 is:

(4, 4)

(1, 1) (6, 6)

(3, 3) (7, 7) (8, 8) (9, 9)

(11, 10) (14, 12)

Note that since all the x_i's in Z_l come from the same Y_a, the $rank_i$ values are distinct and in the range [1, 3w]. To find the element with median rank, we perform a binary search as in Figure 3. For our example, the *while* loop of Figure 3 is iterated 3 times. A trace of Figure 3 is given below:

iteration	low	high	mid	sum
1	1	12	6	3
2	6	12	9	6

3 6 9 7 4

On termination, $mid = 7$ and the element with this rank is 7. So, the median is 7.

Complexity Analysis

On a CREW PRAM, each Y_a can be formed in $O(1)$ time using $O(W)$ processors. So, all Y_a's can be formed in parallel in $O(1)$ time using $O(N)$ processors. All Y_a's can be sorted in $O(\log^2 W)$ time using $O(W)$ processors per Y_a for a total of $O(N)$ processors. This can be done using a bitonic sort [BILA86]. Step 2 is easily performed in $O(1)$ time using $O(N)$ processors. Step 3 can be done in $O(\log^2 W)$ time using bitonic sort and $O(N)$ processors. The subsequence sets Z_l can now be identified in $O(\log W)$ time using 1 processor for each l, $1 \le l \le N$. To find the median of each Z_l, $O(\log W)$ iterations of the *while* loop of Figure 3 are to be performed. The binary search for each subsequence of Z_l can be done in $O(\log W)$ time. So, if we have $O(\log W)$ processors assigned to each Z_l, each iteration of the *while* loop will take $O(\log W)$ time (as each subsequence of Z_l is of size at most W and $|Z_l|$ is $O(\log W)$). The total time for Step 4 is therefore $O(\log^2 W)$ when $O(N \log W)$ processors are available. The processor-time product is $O(N \log^3 W)$.

To compute the MEDIAN1D values for all N rows of I in parallel requires N times as many processors. The run time remains $O(\log^2 W)$ but the processor requirement becomes $O(N^2 \log W)$. Once, MEDIAN1D has been computed, SMEDIAN2D may be computed by applying a similar algorithm using MEDIAN1D as the image. The run time, processor requirement, and processor-time product are unchanged.

4.2 Second Algorithm

As in Algorithm 1, $MEDIAN1D[i, j]$ is computed independently for each row i. The steps in Algorithm 2 are:

Steps1-3: Same as Steps 1-3 of algorithm 1.

Step 4: For $l = 1, \log^2 w + 1, 2\log^2 w + 1, ..$ form the subsequence set Q_l that consists exactly of the $W = 2w + 1$ elements x_c , $nbhd(c, l, w, N)$.

Step 5: Use Q_l to find $MEDIAN[i, l + r], 0 \leq r < \log^2 w$

Consider the same example as for Algorithm 1. While, in Algorithm 1, Z_l is formed for $1 \leq l \leq N = 16$, in Algorithm 2, Only Q_1, Q_5, Q_9, and Q_{13} are formed. Q_9, for instance, consists of elements $x_5 \cdots x_{13}$. The subsequences used are slightly different from those used in Z_9. For Q_l, the subsequences obtained in *Step* 3 are used only to cover the last $W - \log^2 w$ elements of Q_l. The first $\log^2 w$ elements are covered by a new sequence obtained by sorting these $\log^2 w$ elements. So, Q_9 consists of the subsequences:

$$Q_9 = \begin{cases} 3\ 7\ 8\ 9 \\ 5\ 11\ 13\ 14 \\ 10 \end{cases}$$

Let the subsequences in Q_l be $A_0, A_1, \ldots, A_k$ with A_0 being the sequence that contains the first $\log^2 w$ elements of Q_l. Let A_{k+1} be another, initially empty, sequence. Since we shall be deleting from A_0 and inserting into A_{k+1}, we maintain these as balanced binary search trees. $MEDIAN1d[i, l]$ is computed using the algorithm of Figure 3 modified to handle the search trees A_0 and A_{k+1}. In addition, the algorithm computes $S(i)$ and $B(i)$, where $S(i)$ is the largest element in A_i smaller than the median of Q_l and $B(i)$ is the smallest element in A_i bigger than this median. $S(i)$ and $B(i)$ are null if A_i does not contain such elements.

For Q_9 of our example, the median, $MEDIAN(i, 9)$, is 9. So, $S(0) = 8$, $S(1) = 5$, $S(2) = phi$, $S(3) = \phi$, $B(0) = \phi$, $B(1) = 11$, $B(2) = 10$, and $B(3) = \phi$. Now $MEDIAN1D[i, r]$ for $r = 10$, 11 and 12 are computed by successively forming Q_{10}, Q_{11}, and Q_{12}. For Q_{10}, we add $x_{14} = 12$ to A_3 and delete $x_5 = 3$ from A_0 to get:

$A_0 = 7\ 8\ 9$

$A_1 = 5\ 11\ 13\ 14$

$A_2 = 10$

$A_3 = 13$

This requires us to update $B(3)$ to 12. There are four cases in the computation of the new median:

Case 1: The element deleted from A_0 as well as that added to A_{k+1} are smaller than the previous median. In this case the median is unchanged.

Case 2: The element deleted from A_0 as well as that added to A_{k+1} are bigger than the previous median. Here too, the median is unchanged.

Case 3: The element deleted from A_0 is less than or equal to the previous median but that added to A_{k+1} is bigger. The new median is $\min\{B(s) \mid 0 \le s \le k+1 \text{ and } B(s) \neq \phi\}$.

Case 4: The element deleted from A_0 is greater than or equal to the previous median while that added to A_{k+1} is smaller. The new median is $\max\{S(s) \mid 0 \le s \le k+1 \text{ and } S(s) \neq \phi\}$.

For our example, the deleted element is 3 and the inserted element 12. Since the previous median was 9, we are in Case 3 and the new median is $\min\{11, 10, 12\} = 10$. The S's and B's are now updated to:

$S(0) = 9 \quad B(0) = \phi$

$S(1) = 5 \quad B(1) = 11$

$S(2) = \phi \quad B(2) = \phi$

$S(3) = \phi \quad B(3) = 12$

For $MEDIAN1D[i, 11]$ we delete $x_6 = 9$ and insert $x_{15} = 16$ to get:

$A_0 = 7\ 8$

$A_1 = 5\ 11\ 13\ 14$

$A_2 = 10$

$A_3 = 12\ 16$

This causes $B(0)$ to be updated to 8. Since the inserted element is bigger than the previous median and the deleted element smaller, we are again in Case 3. The new median, $MEDIAN1D[i, 11]$, is $\min\{11, 12\}$. The S's and B's are updated and we proceed to compute $MEDIAN1D[i, 13]$ by first deleting x_7 from A_0 and inserting x_{16} into A_3.

Complexity Analysis

For this, we shall assume that only $O(N/(\log W \log\log W))$ processors are available. Using the bitonic sort algorithm of Bilardi and Nicolau [BILA86], Steps 1 through 3 can be done on a CREW PRAM in $O(\log^2 W \log\log W)$ time. For Step 4, we note that $O(N/\log^2 w)$ Q_l's are to be formed. Thus, for each we have $O(\log W/\log\log W)$ processors. Since A_0 initially contains $\log^2 W$

elements, the height of its balanced tree representation is $O(\log\log W)$. A_0 may be set up in $O(\log^2 W \log\log W)$ time using one processor. The remaining A_i's are easily accumulated in $O(\log W)$ time using a single processor. The algorithm of Figure 3 takes $O(\log^2 W \log\log W)$ time to find the median of a Q_l when $O(\log W/\log\log W)$ processors are available to each Q_l. The B's and S's are a byproduct of the searches used to find the median and may now be initialized in $O(\log\log W)$ time. For the remaining medians, each insertion into A_{k+1} and each deletion from A_0 can be done in $O(\log\log W)$ time with a single processor. All the B's and the S's may also be updated in this much time using $O(\log W/\log\log W)$ processors. The *min* of the $O(\log W)$ B's or the *max* of the $O(\log W)$ S's can also be computed in $O(\log\log W)$ time using $O(\log W/\log\log W)$ processors. So the computation of the remaining $\log^2 W - 1$ medians for each Q_l takes $O(\log^2 W \log\log W)$ time. Hence, one row of MEDIAN1D can be computed in $O(\log^2 W \log\log W)$ time using $O(N/(\log W \log\log W))$ processors. All N rows can be computed in this much time using $O(N^2/(\log W \log\log W))$ processors. This algorithm has an optimal processor-time product of $O(N^2 \log W)$.

5 TWO DIMENSIONAL MEDIAN FILTERING

Our parallel algorithm for MEDIAN2D is an adaptation of Algorithm 1 for MEDIAN1D. The Steps are:

Step1: Partition the $N \times N$ image into $w \times w$ subimages $S_{i,j}$ as in Figure 3. For each $S_{i,j}$ form $Y_{i,j} = \bigcup_{\substack{nbhd(i,k,1,N/w) \\ nbhd(j,l,1,N/w)}} S_{k,l}$ and sort Y_{ij}, $1 \le i, j \le N/w$.

Step2: Rank the elements of the $N \times N$ image I. Each element $I[a, b]$ will have nine ranks $rank(y, z, a, b)$, $y \in$ {top row, middle row, bottom row }, $z \in$ {left column, middle column, right column}. Let $S_{c,d}$ be the subimage that contains $I[a, b]$. $rank(y, z, a, b)$ is obtained by looking at each of the nine $Y_{i,j}$'s that contain $S_{c,d}$. y and z are defined by the position of $S_{c,d}$ in $Y_{i,j}$ (see Figure 3) and *rank* is the position of $I[a, b]$ in the sorted $Y_{i,j}$.

Step3: Each $S_{i,j}$ may be tiled with rectangles of size $2^f \times 2^g$, $0 \le f, g \le \lfloor \log_2 w \rfloor$. When w is

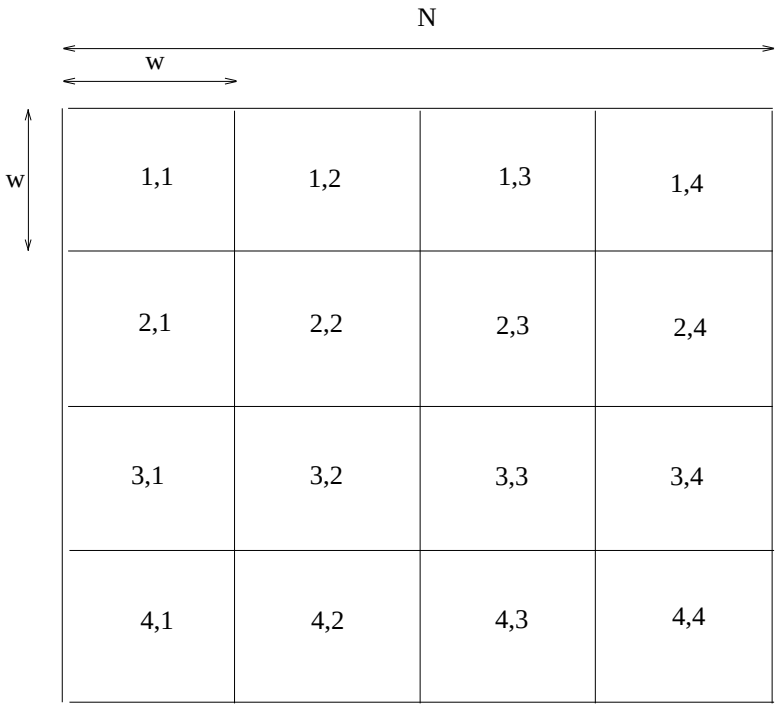


Figure 3: Partitioning of $N \times N$ image into $w \times w$ subimages

not a power of 2, some partial tiles may be used at the ends. Figure 5 shows the tiling for the case $w = 7$.

Step4: For each pair (i, j), $1 \le i, j \le N$ form the sorted sequence set, $U_{i,j}$, from Step 3 that includes exactly the elements $\{I[a, b] \mid nbhd(i, a, w, N)$ and $nbhd(j, b, w, N)\}$

Step5: Search $U_{i,j}$ for the median element. This is $MEDIAN2D[i, j]$.

Complexity Analysis

For the analysis, we assume that $O(N^2 \log^2 W)$ processors are available. Each Y_{ij} of step 1 has $O(W^2)$ elements. As there are N^2/W^2 Y_{ij}'s, we can allocate $O(W^2)$ processors to sort each. This sort can be done in $O(\log^2 W)$ time with this many processors. The ranking of step 2 can now be done in $O(1)$ time. The step 3 sorting may be done by repeatedly merging smaller tiles to get bigger ones. In each merge stage pairs of tiles of some fixed size are merged to get sorted tiles of the next size. There are $O(\log^2 W)$ tile sizes and $O(\log W)$ merge stages each taking $O(\log W)$ time are needed (Figure 6). So, step 3 is done in $O(\log^2 W)$ time with the given number of processors.

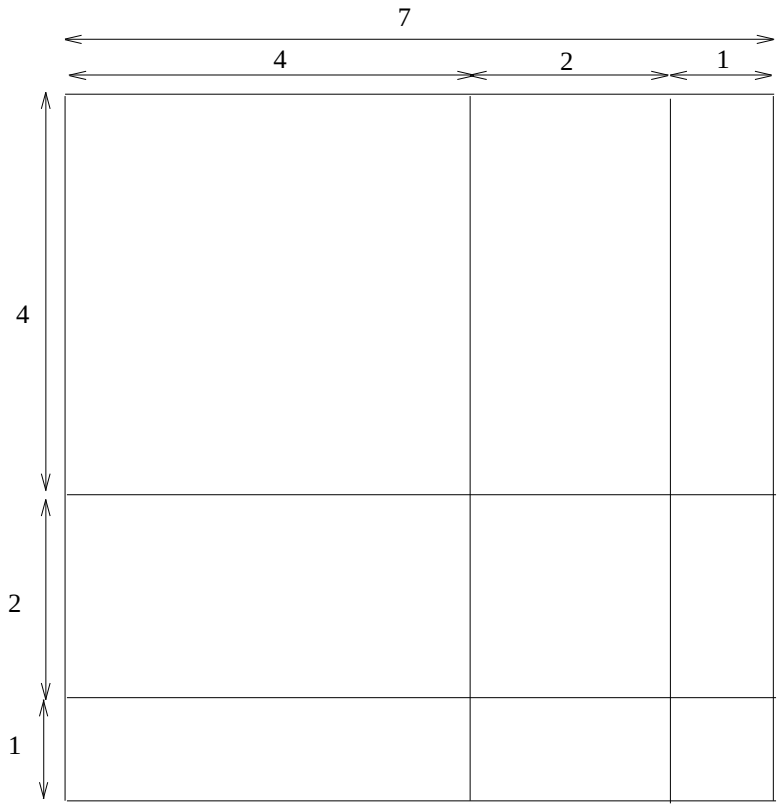
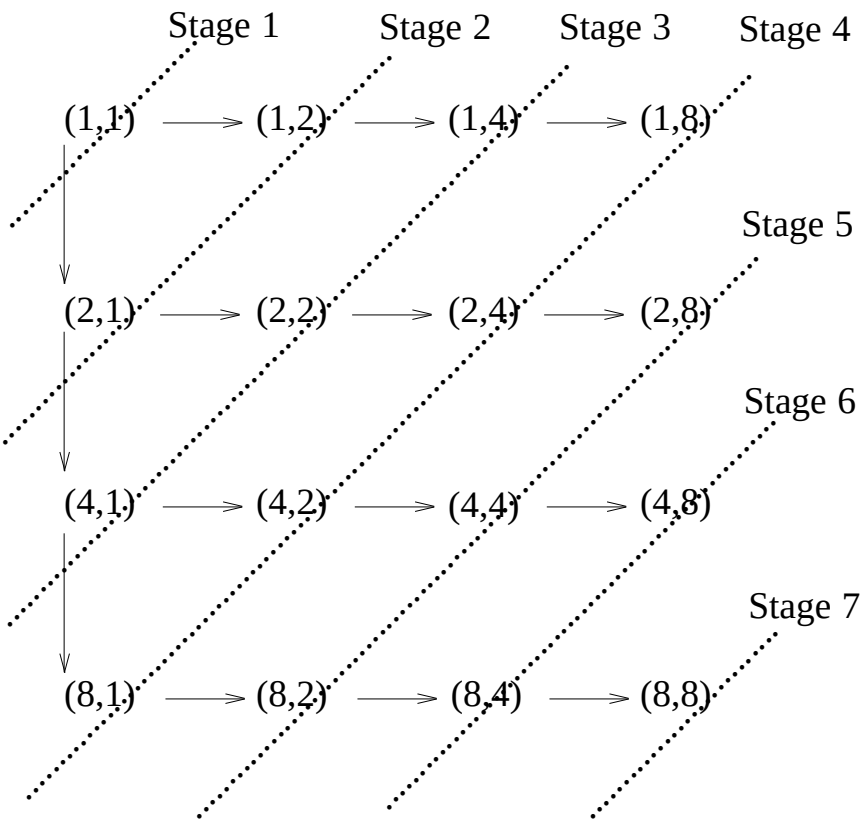


Figure 5: Tiling of $S_{i,\,j}$ (w = 7)

For step 4, 1 processor is used to form each U_{ij}. Each U_{ij} can be formed so as to have only $O(\log^2 W)$ sorted sequences. Each such sequence is at most $O(W^2)$ long. The step 4 time is $O(\log^2 W)$. The step 5 search of each U_{ij} is done using an algorithm almost identical to that of Figure 3 (the rank range needs to be increased from $3w$ to $9w^2$). Using $O(\log^2 W)$ processors for each U_{ij}, this search takes $O(\log^2 W)$ time. So the total time for the algorithm is $O(\log^2 W)$ and the processor requirement is $O(N^2 \log^2 W)$. The processor-time product is $O(N^2 \log^4 W)$ which is better than that for the serial algorithm that uses balanced search trees.

6 CONCLUSION

We have developed an algorithm for separable median filtering that requires at most 2 comparisons per median element when the window size is 3. This represents a theoretical improve-



... represent the stages
---> represent the dependencies
Figure 6: Stages required in Merging

ment over the result of [BASU87]. Our algorithm, however, runs only marginally faster that of [BASU87]. Fast parallel CREW PRAM algorithms with good processor-time product have also been developed for separable median filtering and two dimensional median filtering. The following problems remain open:

(1) Is there an $O(\log N)$ CREW PRAM algorithm with optimal processor-time product for Separable Median Filtering?

(2) What is the complexity of an optimal serial algorithm for two dimensional median filtering?

(3) Find an $O(\log N)$ CREW PRAM algorithm for two dimensional median filtering with good processor-time product (within a polylogarithmic factor of the optimal).

7 REFERENCES

[AJTA86] M. Ajtai, J. Komlos, W. L. Steiger, and E. Szemeredi, "Deterministic Selection in $O(\log\log N)$ Parallel Time", *Proceedings of 18th ACM Symposium on Theory of Computing*, **1986**, pp. 188-195.

[BALL85] D. H. Ballard and C. M. Brown, "*Computer Vision*", **1985**, Prentice Hall, New Jersey.

[BASU87] A. Basu and C. M. Brown,"Algorithms and Hardware for Efficient Image Smoothing", *Computer Vision, Graphics and Image Processing*", Vol. 40, **February 1987**, pp. 131-146.

[BILA86] G. Bilardi and A. Nicolau, "Bitonic sorting with $O(N\log N)$ comparisons", *Proceedings of 1986 Conference on Information and System Sciences*, Department of Electrical Engineering, Princeton University, pp. 366-341.

[JAYA76] N. S. Jayant, "Average and median based smoothing techniques for improving digital speech quality in the presence of transmissions error", *IEEE Trans. on Communication*, Vol. COM-24, **September 1976**, pp. 1043-1049.

[HORO86] E. Horowitz and S. Sahni, "*Fundamentals of Data Structures in Pascal*", Computer Science Press, Maryland, **1986**.

[HUAN79] T. S. Huang, G.Y. Yang, and G. Y. Tang, "A Fast Two dimensional Median Filtering Algorithm", *IEEE Transaction on ASSP*, Vol. ASSP-27, No. 1, **February 1979**, pp. 13-18.

[NARE81] P.M. Narendra. "A Separable Median Filter for Image Noise Smoothing", *IEEE Trans. on PAMI*, Vol. PAMI-3, No. 1, **January 81**, pp. 20-29.

[RABI75] L. R. Rabiner, M. R. Sanbur and C. E. Schmidt, "Applications of a non-linear smoothing operation to speech processing," *IEEE Transaction on ASSP*, Vol. ASSP-23, **December 1975**, pp. 552-557.

[SHAM76] M. I. Shamos, "Geometry and Statistics: Problems at the Interface", *Algorithms and*

Complexity: New Directions and Recent Results, Edited by J.F. Traub, **Academic Press Inc.**, New York, **1976**.

[STOU83] Q. Stout, "Sorting, Merging, Selecting and Filtering on Tree and Pyramid Machines", *Proceedings of 1983 International Conference on Parallel Processing*, pp. 214-221.

[TANI82] S. L. Tanimoto, "Sorting, histogramming, and other statistical operations on a pyramid machine", *Department of Computer Science Report 82-08-82, University of Washington, Seattle*.