International Journal of MC Square Scientific Research Vol.1, No.1,2009

High Speed Pipelined Architecture for Adaptive Median Filter

A. Mohamed Mian[1]
Assistant Professor Assistant Professor Department of ECE
Abdul Hakeem College of Engineering and Technology
Vellore, India

Abstract

Low level data processing functions, like FIR filtering, pattern recognition or correlation, where the parallel implementation is supported by architecture matched special purpose arithmetic; high throughput FPGA circuits easily outperform even the most advanced DSP processors. In this paper investigates a high-speed non- linear Adaptive median filter implementation is presented. Then Adaptive Median Filter solves the dual purpose of removing the impulse noise from the image and reducing distortion in the image. Adaptive Median Filtering can achieve the filtering operation of an image corrupted with impulse noise of probability greater than 0.2.

Keywords: Salt and Pepper, Adaptive, Pipeline, FIFO, Rank order, Non linear.

1. Introduction

Median filtering is a powerful instrument used in image processing. The traditional median filtering algorithm, without any modifications gives good results. There are many variations to the classical algorithm, aimed at reducing computational cost or to achieve additional properties[1]. Median filters are used mainly to remove salt-and pepper noise. Doing this, they preserve edges in the image (preserve their location and do not affect their steepness, unlike Gaussian filters), but unfortunately median filtering may destroy small features in the image[2]. A way to avoid it is to apply center- weighted median filtering instead of a plain median, but the drawback of this solution is the detoriation of the filter's ability to suppress impulse noise[3].

Common drawback of various kinds of the median filtering is their computational cost[4]. Computing a two-dimensional median for an NxN window, requires sorting of NxN elements for every image pixel and choosing the median value for the output[5]. After the sorting, each queue element is assigned a value called a rank, specifying its position in the queue as a result of sorting[6].

Therefore, using median filtering in any real- time vision system requires a significant computational power. One way to speed up the computations is to implement the algorithm in hardware, e.g. with the help of FPGA circuits. The rationale behind this is to use the FPGA's inherent ability to execute operations in parallel. Moreover, programmable logic creates the possibility to tailor the implementation to the user's needs. All these results are in a significant speedup over the software implementations by using sequential processors. One drawback of hardware-based algorithm development is the complexity of the design process as implementing algorithmically complex operations is very difficult. Median filtering, like many other low-level image processing algorithms is fairly simple - the main problem in this case is the amount of data to handle.

2. Median Filter

Median filtering is a non-linear, low-pass filtering method, which you use to remove "speckle" noise from an image. A median filter can outperform linear, low-pass filters on this type of noisy image because it can potentially remove all the noise without affecting the "clean" pixels. Median filters remove isolated pixels, While a hardware system and a matrix-manipulating software program are fundamentally different, they can produce identical results, provided that care is taken in development. This approach was taken because it speeds understanding of the algorithm design. In addition, this approach facilitates comparison of the software and synthesized hardware algorithm outputs. This project is focused on developing hardware implementations of image processing algorithm for use in an FPGA based image processing system. The rank order filter is a particularly common algorithm in image processing systems.. For every pixel in an image, the window of neighboring pixels is found. Then the pixel values are sorted in ascending, or rank, order. Next, the pixel in the output image corresponding to the origin pixel in the input image is replaced with the value specified by the filter order. The VHDL code can be simulated to verify its functionality. The Implementation of a 3 × 3 filter window is shown in figure 1.

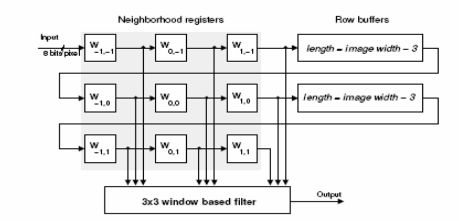


Figure. 1Implementation of a 3 × 3 filter window

3. Adaptive Median Filter

The Adaptive Median Filter is designed to eliminate the problems faced with the standard median filter. The basic difference between the two filters is that, in the Adaptive Median Filter, the size of the window surrounding each pixel is variable. This variation depends on the median of the pixels in the present window. If the median value is an impulse, then the size of the window is expanded. Otherwise, further processing is done on the part of the image within the current window specifications. 'Processing' the image basically entails the following: The center pixel of the window is evaluated to verify whether it is an impulse or not. If it is an impulse, then the new value of that pixel in the filtered image will be the median value of the pixels in that window. If, however, the center pixel is not an impulse, then the value of the center pixel is retained in the filtered image. Thus, unless the pixel being considered is an impulse, the gray-scale value of the pixel in the filtered image is the same as that of the input image. Thus, the Adaptive Median Filter solves the dual purpose of removing the impulse noise from the image and reducing distortion in the image. Adaptive Median Filtering can handle the filtering operation of an image corrupted with impulse noise of probability greater than 0.2. This filter also smoothens out other types of noise, thus, giving a much better output image than the standard median filter.

4. Moving Window Architecture

In order to implement a moving window system in VHDL, a design was devised that took advantage of certain features of FPGAs. FPGAs generally handle flip -flops quite easily, but instantiation of memory on chip is more difficult. Still, compared with the other option, off-chip memory, the choice using on-chip memory was clear. It was determined that the output of the architecture should be vectors for pixels in the window, along with a data valid signal, which is used to inform an algorithm using the window generation unit as to when the data is ready for processing. Since it was deemed necessary to achieve maximum performance in a relatively small space, FIFO Units specific to the target FPGA were used. Importantly though, to the algorithms using the window generation architecture, the output of the window generation units is exactly the same. This useful feature allows algorithm interchangeability between the two architectures, which helped significantly, cut down algorithm development time. A window size was chosen because it was small enough to be easily fit onto the target FPGAs, and is considered large enough to be effective for most commonly used image sizes. With larger window sizes, more FIFOs and flip -flops must be used, which increases the FPGA resources used significantly. Figure 2, 3 shows a graphic representation of the FIFO and flip flop architecture used for this design for a given output pixel window.

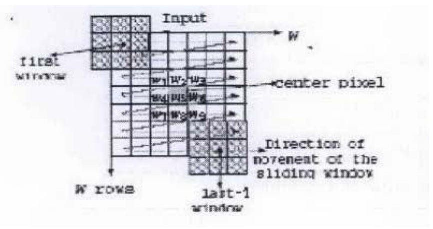


Figure.2 Moving Window Architecture

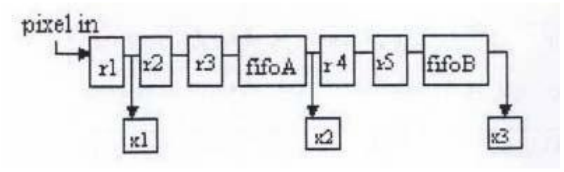
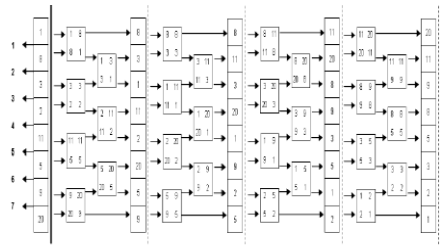


Figure.3 Reading Pixels from Window

5. Parallel Sorting Strategy

To make a fair comparison of the parallel sorting strategy against wave sorter strategy in terms of the total number of required steps to sort an array, it is necessary to consider the steps used to read data from memory and the steps required to store the sorted data back to memory. The proposed approach is based on the same structure of the registers array used in the wave sorter strategy. With this kind of array, data can be stored in the array by sending a datum to the first register and later, when the second datum is sent to the first register, the value on the first array is shifted to the second register. If k sorts are required, then the parallel sorting requires to ((n+n/2) * k + n) to sort an array of n data. The figure 4 shows the parallel sorting with two levels of comparators performance. Thus total number of steps required can be obtained by the following equation:



$$f_{steps}^{PS}(n,k) = n\left(\frac{3}{2}k+1\right)$$

Figure. 4 Parallel sorting with two levels of comparators performance

The parallel strategy leads to a significant reduction compared to the wave sorter approach. Furthermore, in additional sorts the necessary number of steps for sorting is equal to the number of characters in the biggest group of identical characters divided by 2 (remember that an additional sorting is implied if groups of identical adjacent characters appear in the array).This implies that in practice, it is possible to reduce more than the number of steps to solve the suffix problem.

6. Implementation and Testing

The adaptive filter works on a rectangular region Sxy. The adaptive median filter changes the size of Sxy during the filtering operation depending on certain criteria as listed below. The output of the filter is a single value which the replaces the current pixel value at (x, y), the point on which Sxy is centered at the time. The following notation is adapted from the book and is reintroduced here:

Zmin = Minimum gray level value in Sxy.
Zmax = Maximum gray level value in Sxy
Zmed = Median of gray levels in Sxy
Zxy = gray level at coordinates (x, y)
Smax = Maximum allowed size of Sxy
The adaptive median filter works in two levels denoted Level A and Level B as follows:

```
Level A:        A1= Zmed - Zmin
                A2= Zmed – Zmax
 If A1 > 0 AND A2 < 0, Go to
Level B
Else increase the window size
If window size <= Smax repeat
Level A
 Else output Zxy.
Level B:
          B1 = Zxy – Zmin
           B2 = Zxy – Zmin
If B1 > 0 and B2 < 0 output Zxy Else output Zmed.
```

7. Result

The adaptive median filter is designed to remove impulsive noise from images. Therefore, our algorithm's performance was first tested with basic salt and pepper noise with a noise density of 0. 25. The next test involves processing images that contain impulsive and/or non- impulsive noise. It is well known that the median filter does not provide sufficient smoothening of non-impulsive noise. Therefore, Gaussian and 'salt and pepper' noise were added to the image which was then processed by the algorithm. The Figure 5 shows the performance of the adaptive median filter.

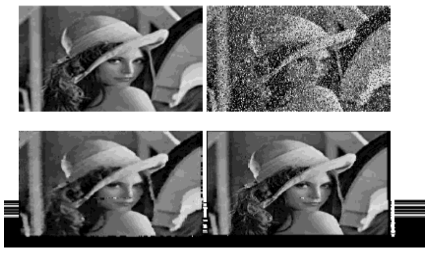


Figure. 5 Results of filtering with a 3*X*3 median and conditional median filter. From left to right, first row: original Image, noisy image; second row: standard median filter, Adaptive median filter.

8. Conclusion

The architecture is pipelined which processes one pixel per clock cycle, thus to process an image of size 256 x 256 it requires 0.65 ms when a clock of 100 MHz is used and hence is suitable for real time applications The adaptive median filter successfully removes impulsive noise from images. It does a reasonably good job of smoothening images that contain non-impulsive noise. Overall, the performance is as expected and the successful implementation of the adaptive median filter is presented.

References

1. Zdenek Vasicek, Lukas Sekanina, “Novel Hardware Implementation of Adaptive Median Filters.” IEEE.2015.

2. Olli Vainio, Yrjö Neuvo, Steven, and E. Butner, “A Signal Processor for Median-Based Algorithms”, IEEE Transactions on Acoustics, Speech, Processing, vol .37, no.9, 1989.

3. Bapeswara Rao, V.V, and K. Sankara Rao, “A New Algorithm for Real-Time Median Filtering”, IEEE Transactions on Acoustics, Speech, Processing, vol. 34, no. 6, 1986.

4. Ahmad, M. O, and D. Sundararajan, “Parallel Implementation of a Median Filtering Algorithm”, Int. Symp. on Signals and Systems, 1988.

5. Dobrowiecki Tadeusz, Medián Szűrők, and Mérés és Automatika, 37. Évf., 1989. 3.szám

6. Xilinx Foundation Series Quick Start Guide, 1991-1997. Xilinx. Inc