A Novel Technique for Mean Filter Design and Implementation via Systolic Array

Asmaa Hameed Rasheed
College of Engineering, Baghdad University, Baghdad, Iraq

Abstract: Mean filter is widely used with image processing such as image smoothing or denoising system. The image is usually scanned by windowing technique and all the pixels of the specified window are drawn in whole and processed together in each cycle. This process do repeated calculations and use more hardware that can be reduced highly if an efficient implementation method is used. Another important problem with traditional implementation method of this filters, it enters all the pixels of the processed window entirely in each time that need more I/O pins. These I/O pins act as a big limitations that can restrict any huge data applications when implemented with high speed H/W such as FPGA kits. In this study, two novel techniques are proposed to design and implement mean filtering process in which the hardware complexity can be resolved and efficiently improve processing time as well as the input pins is minimized highly by taking only the new pixels of the processed window while keeping the other big parts that are already prepared in previous cycles. The proposed techniques depend mainly on removing the repeated computation to reduce the numbers of addition as well as cancellation of division operations which result in improving the processing efficiency and highly reduce execution time to match real time requirements. Pipelining technique is also adopted to activate parallel processing scheme. So, systolic architecture are used for further reduction in computation operations to improve overall system execution time. The proposed design techniques has been tested on a variety of grey images having different sizes. The obtained results proved good improvements in processing time with increasing image size. The efficiency of the implemented design is improved significantly with large filter length that make the proposed technique very useful when dealing with huge images.

Key words: Mean filtering, noise reduction, spatial filters, systolic array, FPGA’s, system generator

INTRODUCTION

Most image processing depend mainly on spatial filters that done by convolution masks. Image denoising, sharpening, blurring, edge detection and many other image processing necessitate using of masks. The mean filter is one of the most important masks that extensively used in different image processing (Jain, 1989).

Image filtering is the more important part in any image denoising system. The filtering process can be accomplished in frequency domain or in many cases, time domain filters can be very useful and mostly used for variety of image applications. Different spatial filters are used with image denoising system depending on the corrupted noise (Gonzalez and Woods, 2007).

If a Gaussian noise corrupt the original image, then mean filter is the most suitable one for noise removing in such cases. The efficiency of mean filter for noise removal with Gaussian noise refers to the fact that Gaussian noise values are concentrated around the mean as illustrated in Fig. 1 which shows the probability density function of Gaussian noise (Boneclet, 2000).

When large images is considered to be processed and large window sizes is dependent then any application need these filters can be quite time consuming. For example, the basic implementation of a mean filter for a 256×256 image with a 3×3 window size, needs more than 5×10² additions and also more than 65×10³ divisions. We must exploit and assign the inherent redundancies in this traditional implementation method.

Mean filter: Mean filter is a linear digital filter that use a mask window of odd size to scan the image horizontally and vertically. The windowing operation is illustrated in Fig. 2.
Fig. 2: Windowing technique

Smoothing is done by reducing the variation in values for successive pixels. The smoothing operation is in fact equivalent to low-pass filtering that carefully smooths any sudden change of pixel value by altering its value by average (mean) of its surrounding pixels. Smoothing filters or in more precisely mean filtering has a big area in noise removal and blurring of image processing (Kumar and Kumar, 2015).

Mean filtering is inherently a convolution operation as the mask is scanned the image until every pixel has been processed. Kernel is the base of convolutions process which is the shape and size of the surrounding neighbors that used for mean calculation. Large kernels are needed when smoothing must be deeply (Zhang and Li, 2014).

The traditional method of mean filtering implementation is very simple but inefficient due to redundant and highly re-computations of the accomplished additions in the previous mask. The required number of additions is \( U^*V*n^2 \) and the needed division operations is \( U^*V \) only. Thus, the number of additions and divisions are related directly to the image size, so, the problem of high consuming time appear clearly with large images which is the important factor for real time processing.

A major redundancy caused by re-computation of the sum of all pixel values of the specified window when the mask is moved from specified pixel to the successive one. As the mask is shifted by one pixel the sum of the new column in the mask window is needed only, since, the sum of all other columns is already computed for the previous pixel.

Mean filter implementation: As shown in Fig. 2, the window of size \( n^*n \) firstly, scan the image horizontally and when the processed raw of the specified image is completed, then it begins a vertical path to process the second raw of the noisy image and so on.

\[
\begin{array}{|c|c|c|}
\hline
\text{Before filtering} & \text{After filtering} \\
\hline
3 & 3 \\
4 & 4 \\
7 & 5 \\
\hline
\end{array}
\]

Fig. 3: a, b) Mean filtering process

The mean filter process is in fact computing the average (mean) of all pixels of the specified window. So, a summation of all pixels of the specified window must be obtained. Then use the obtained sum to compute the average of the current mask window to be replaced with the value of the centered pixel in the processed window. So, after finding the summation of all pixels in the specified windows, the mean of these pixels must be computed. Then the filter replaces the value of the center pixel with the mean (average) of all the pixel values in the window. The window usually have a square shape but it can be any other shape that has a symmetry around its center. Figure 3 shows an example of \( 3^*3 \) mean filter (taking specific window). The summation of all values in the given window is:

\[
4+7+3+2+5+9+8+6+1 = 45
\]

The mean (average) value is:

\[
45/9 = 5
\]

The original value of the centered pixel is 9 and after mean filtering it was replaced by the mean of all nine values (Zhang and Li, 2014).

Traditional mean filtering algorithm: The classical method of mean filter implementation can be summarized in the following steps:

\[
\begin{array}{c}
\text{for every pixel in the image do} \\
\text{sum all the values in the mask} \\
\text{find the average of these values} \\
\text{replace the pixel value with mean value} \\
\end{array}
\]

Hardware implementation of mean filter: The ordinary method for hardware implementation of mean filters is accomplished by scanning the window of specified size \( (n^*n) \) over the entire image horizontally and vertically. The number of pixels of each window is equal to \( n \) multiplied by \( n(n^*n) \) pixels. All these pixels are entered simultaneously in parallel and summed together to compute the average of them. For \( n^*n \) window size it is
Fig. 4: Traditional hardware implementation of mean filter

needed \((n^2-n-1)\) adder and need \(n^2n\) input pins from the recommended hardware that used as an implementation platform. Figure 4 shows traditional hardware implementation of 3*3 mean filter and it required eight adders. It is clear the number of required division operations in each step is only one.

**MATERIALS AND METHODS**

**Proposed technique:** The proposed technique depends on the fact that the new window is highly similar to the previous one (for 3*3 mask window, similarity of 66.6%) and differs from it by only one column (also for 3*3 mask window, difference ratio is 33.3%). Then there is no need to enter all the pixels of the new window but only the column that represent the difference part between the previous and next windows should be entered. As well as the summation of the big parts of the new window need not to be calculated next time, since, it is exist from the summation process of the previous window. In this way, the number of required adders will be reduced from \((n^2n-1)\) in the traditional technique to only \((2^n(n-1))\) with proposed techniques.

The proposed techniques deal with clock sequencing to decide the suitable position of the input column. If mod (clock \(seq/3\)) = 1 then assign entered column 1 to first stage:

- If mod(clock \(seq/3\)) = 2 then assign entered column 2 to first stage
- If mod(clock \(seq/3\)) = 0 then assign entered column 3 to first stage

The hardware implementation of this improved method is well illustrated in Fig. 5.

**Proposed systolic technique:** Systolic architecture are used for further reduction in addition operations. This proposed approaches convert the concurrent processing to sequential form as a pipelined stages, each stage responsible on part of the complete job and deliver its output to the next stage synchronized by a system clock (Wan, 1996). The number of additions can be minimized using pipelining technique, to only \((n-1)\) addition operation. The number of input pins is also minimized from \((n^2n)\) pins to only \((n)\) input pins with proposed techniques as shown in Fig. 6. The 5*5 mean filter design based on systolic array has been shown in Fig. 7.

**Avoidance of division operation:** In this research, we want to simplify the structure of hardware implementation of mean filtering process via the assistance of systolic array. The hardware simplification process start by minimizing the number of used adders to find the summation of all pixels values within the specified window. Now, another try has been made to avoid the division operation that is essential to compute the mean (average) from the obtained sum. The division process as it obvious, need a complicated hardware, so an alternate method is used to avoid this division process.

The pixel values of the gray image can have only integers values that varied from 0-255 for an 8-bit grayscale and refuse any value with floating point format but the division result can be floating point that mismatch the image coding and representation form so rounding the result must be done to convert it to an integer value. Figure 8, explains the process of determining the average, assume the average is \(X\), from different expected sum’s. Figure 8 shows that when having more than the half numbers of pixels for a specified value \(X\) (the used \(X\) value, here is 125) and the other values is for the previous or next value then all these sum’s will be considered to have an average of \(X\).

If the summation of the specified pixels vary between 1121-1129 then the average can be swing from 124.5-125.4 with step of 0.1 and all the values in this range can be considered as an integer of 125, since, any pixel value with floating point format can not be accepted for gray
Fig. 5: Proposed hardware implementation of mean filter

Fig. 6: Systolic hardware implementation of mean filter

Fig. 7: Traditional hardware implementation of mean filter

Fig. 8: a, b) Expected mean for different sums without division operation
image. And when the sum vary between 1130-1138 the
dependant average is 126 while when the sum is between
1112-1120, the average would be 124 and so on as
illustrated in Fig. 8b.

**FPGA based design:** The proposed technique for mean
filtering process using systolic array has been
implemented using FPGA kit, Spartan 3-700 A with
cooperative of MATLAB package system generator
technique (Base, 2007; AlAli et al., 2013; Jeon et al.,
2013). Xilinx Software ISE 14.7 and its matched version
of MATLAB package R2013a are dependant in the
present research with aid of system generators block
sets.

The hardware implemented system for mean filtering
based on FPGA is shown in Fig. 9. The black box was
programmed by VHDL language (Perry, 2002) to
implement the circulating switching as follows:

```
PROCESS (clk)
BEGIN
  IF rising_edge (clk) THEN
    i := i+1
    IF i = 1 THEN
      C1 <= SUM1
    END IF
    IF i = 2 THEN
      C2 <= SUM1
    END IF
    IF i = 3 THEN
      C3 <= SUM1
      i <= 0
  END IF
END PROCESS
```

**RESULTS AND DISCUSSION**

The two proposed methods were tested for images of
8-bit gray-level and these images have different sizes
starting by 64*64 and ending with image size of
1024*1024. Each image was applied for three different
methods:

- Traditional method
- Proposed method
- Developed method using systolic array architecture

For all these methods the testing is considered for
varying masking window size sewing from 3*3-11*11
window size. The device utilization summary for the
adopted three methods are show in Table 1-3, respectively.

To declare the comparison among the three tested
methods, the used numbers of adders were computed to
single window for each method with respect to different
sizes of masking window vary from 3*3 and ending with
11*11 window size as illustrated in Table 4. Different
images with variable size ranged from 64*64-1024*1024
and repeat the above calculation for each image that well
illustrated in Table 5-7.
Table 3: Utilization summary of FPGA based systolic H/W design technique

<table>
<thead>
<tr>
<th>Device utilization summary</th>
<th>Used</th>
<th>Available</th>
<th>Utilization (%)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of 4 input LUTs</td>
<td>15</td>
<td>11.779</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>No. of occupied slices</td>
<td>9</td>
<td>5.888</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>No. of slices containing only related logic</td>
<td>9</td>
<td>56</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>No. of slices containing unrelated logic</td>
<td>0</td>
<td>56</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Total No. of 4 input LUTs</td>
<td>16</td>
<td>11.776</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>No. used as logic</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. used as a route-thru</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of bonded IOBs</td>
<td>372</td>
<td>13</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>IOB flip flops</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of BUF/GMUXs</td>
<td>24</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>No. of MULT/18X18SIOs</td>
<td>20</td>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Average waittime of non-clock nets</td>
<td>1.44</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: The No. of required adders according to window size for different implementing techniques

<table>
<thead>
<tr>
<th>Used methods/window sizes</th>
<th>Traditional methods</th>
<th>Proposed methods</th>
<th>Proposed methods with Systolic array</th>
</tr>
</thead>
<tbody>
<tr>
<td>3*3</td>
<td>8</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5*5</td>
<td>24</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>7*7</td>
<td>89</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>9*9</td>
<td>80</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>11*11</td>
<td>120</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 5: Approximated No. of used adders for different image size with different design techniques

<table>
<thead>
<tr>
<th>Used methods/image sizes</th>
<th>Traditional methods</th>
<th>Proposed methods</th>
<th>Proposed methods with Systolic array</th>
</tr>
</thead>
<tbody>
<tr>
<td>64*64</td>
<td>33*10^3</td>
<td>59*10^3</td>
<td>20*10^3</td>
</tr>
<tr>
<td>128*128</td>
<td>13*10^3</td>
<td>40*10^3</td>
<td>79*10^3</td>
</tr>
<tr>
<td>256*256</td>
<td>53*10^3</td>
<td>16*10^3</td>
<td>26*10^3</td>
</tr>
<tr>
<td>512*512</td>
<td>2*10^4</td>
<td>6*10^3</td>
<td>12*10^3</td>
</tr>
<tr>
<td>1024*1024</td>
<td>9*10^4</td>
<td>25*10^4</td>
<td>50*10^4</td>
</tr>
</tbody>
</table>

Table 6: Utilization summary of FPGA based improved H/W design technique

<table>
<thead>
<tr>
<th>Techniques/operators</th>
<th>Traditional methods</th>
<th>Improved methods</th>
<th>Systolic methods</th>
<th>Advanced methods</th>
<th>Advanced systolic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adders</td>
<td>n*n-1</td>
<td>2*(n-1)</td>
<td>n-1</td>
<td>n</td>
<td>n+1</td>
</tr>
<tr>
<td>Dividers</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 7: Processing time for 1024*1024 image size with different techniques

<table>
<thead>
<tr>
<th>Used methods/window sizes</th>
<th>Traditional methods</th>
<th>Proposed methods</th>
<th>Proposed methods with Systolic array</th>
</tr>
</thead>
<tbody>
<tr>
<td>3*3</td>
<td>0.6352768</td>
<td>0.0182044</td>
<td>0.010113</td>
</tr>
<tr>
<td>5*5</td>
<td>0.055945</td>
<td>0.019980</td>
<td>0.011100</td>
</tr>
<tr>
<td>7*7</td>
<td>0.080372</td>
<td>0.021159</td>
<td>0.011750</td>
</tr>
<tr>
<td>9*9</td>
<td>0.125998</td>
<td>0.026249</td>
<td>0.014503</td>
</tr>
<tr>
<td>11*11</td>
<td>0.183752</td>
<td>0.031681</td>
<td>0.017600</td>
</tr>
</tbody>
</table>

The varying number of adders for each image by applying all the implemented methods in the present research are shown in Fig. 10 while Fig. 11 repeat the same graph but for the image of size 1024*1024 only.

The increasing number of adder according to window size for the three tested methods is will illustrated in plot of Fig. 12 and the inverse plot that shows the number of adders with respect to these implementing methods against different window size is shown in Fig. 13 that applied to largest image also. The plot of execution time against varying window size for the largest test image which have a size of 1024*1024 is shown in Fig. 14.

![Fig. 10: The No. of adders with respect to varying image size for different techniques](image_url)
Fig. 11: The No. of adders with respect to different techniques for 1024*1024 image size

Fig. 12: The No. of adders with respect to different techniques for varying window size for 1024*1024 image size

Fig. 13: The No. of adders with respect to varying window size with different techniques for 1024*1024 image size

The plots that shows the big arising in number of adders with respect to image size for different window size are shown in Fig. 15-17 for traditional, improved and systolic method, respectively. Figure 18 shows the tested images which are chosen to be varied in image details complexity. Thus, image 1 is very simple with big flat area of approximately same gray level while more details appeared in image 2 and these precise details be more complicated in image 3 and so on to reach maximum complexity in image 5.
Fig. 17: The No. of adders with respect to varying image size for systolic technique according to different window size

(a) (b) (c)

Fig. 18: Continue
CONCLUSION

Two different methods are invented in the present research to minimize highly hardware design complexity and in the same time improve the execution time to match real time requirements that was found to be efficiently reduced, especially for large images.

The two proposed methods have been implemented using the field programmable gate array and they are found to be area efficient. The improved method utilize only 7% from the available hardware component in the FPGA kit while the utilized area for proposed systolic method is 3% only. This reduction in hardware complexity is due to the efficient minimizing in the required numbers of adders for each method. The proposed improved method reduces the used adders from (n*n-1) to only 2*(n-1) and the proposed systolic technique consumes as less as (n-1) adders. If a storage element is used to keep the previous calculated sum in each step then further reduction can be gained to get (n) adders for the proposed improved method.

The division operations can be canceled in the present research that efficiently simplifies the hardware design as well as reducing processing time. This cancellation is done by storing the expected integer results of division for different ranged sums.

The proposed methods presented in this research have successfully improve the execution time, especially, for large images. The implemented systems are tested for different image sizes with varying masking window sizes ranged from 3*3 to 11*11 and the obtained results and plots show excellent improvements in both hardware minimization as well as speed up processing time that encourage to depend the proposed techniques for many real time applications related to different image processing.

The filtered images which differ highly in their complicated characteristic details, show that the mean filter behaves better for images with complicated details than for simple ones.

REFERENCES


