Evaluation of Noise Reduction Filters in Medical Image Processing using OpenMP

Luis Cadena, Darwin Castillo, Aleksandr Zotin, Franklin Cadena, Patricia Diaz

Abstract—Most of the medical diagnostic requires studies of medical images for to give an accurate treatment. Therefore is important the improvement of the medical images in terms of noise, quality, and morphological definition. The medical images series contains approximately 20% of noise caused by the equipment itself, especially in the x-ray modality.

Therefore, the principal aim of this project is to develop an algorithm that helps suppress the noise as a preprocessing stage before medical analysis. The algorithm for noise reduction proposed uses classic and optimized mean filter, Gaussian filter, and median filter.

This project also uses OpenMP parallel programming to optimize processing time and computational resources. The parallel implementation results of algorithms with sequential and classic implementation show great performance in the quality of the time processing, noise localization, and noise reduction. This improvement helps medical professionals get better details about the different pathologies for effective diagnostics and treatment.

Index Terms—medical image, Mean Filter, Median Filter, Gaussian 2D, parallel programming, OpenMP.

I. INTRODUCTION

A medical image is a two-dimensional (2D) distribution of small image points called pixels. Mathematically point view, it can be considered as a function of two real variables, for example, \( f(x,y) \) with \( f \) as the amplitude of the image at position \((x, y)\) [1-3]. In the last years, image processing has attracted the attention of multidisciplinary fields such as applied mathematics, computer sciences, engineering, statistics, physics, biology, and medicine.

Medical imaging is the technique and process used to create anatomic, physiological or functional images for clinical or medical purposes. There are many different medical image modalities like CT, PET, MRI, X-ray, Ultrasound imaging, fMRI, etc. [2], [4-8].

Computer-aided diagnostic processing has already become an important part of the clinical routine because the medical image processing plays an important role in the diagnosis and detection of the sicknesses and the treatment [4].

For example, in radiology images, the tissues of the human body absorb radiation and the image is projected in different shades of black and white; the bone tissue is observed white, the fat and soft tissues are observed gray and the air is observed black. This type of image is widely used in trauma for find infections, benign or malignant bone lesions, degenerative joint disorders, lung diseases, abdomen and pelvis lesions, mammary glands, and to locate foreign objects and guide procedures [5]. Another type of important medical image is magnetic resonance imaging (MRI) because it provides anatomical and physiological information in a non-invasive way. MRI does not use any kind of ionizing radiation. MRI creates images of structures through the interactions of magnetic fields and radio waves with tissues [6].

Medical images independently of their type are often contaminated by impulsive, additive or multiplicative noise caused by the imaging process and the equipment itself. The noise usually corrupts medical images by replacing some of the pixels of the original image with new pixels having luminance values near or equal to the minimum or maximum of the allowable dynamic luminance range. The noise conditions determine the type of it [3, 19, 20].

Therefore, the principal aim of this work is to develop an algorithm that helps suppress the noise of different anatomical structures of X-ray modality, as a pre-processing stage before medical analysis and diagnostic.

The algorithm for noise reduction uses classic and optimized mean filter, Gaussian filter, and median filter to determine the pixel value in the noiseless image and remove it. Also, this project uses a parallel programming model (OpenMP) [8-10] to optimize the processing time and computational resources. The parallel implementation results of algorithms with sequential and classic implementation show...
great performance in the quality of the time processing, noise localization, and noise reduction.

This improvement helps medical professionals get better details about the different pathologies for effective diagnostics and treatments.

Applications of Parallel Computing are: Bioscience, Biotechnology, Genetics, Medical imaging and diagnosis, Chemistry, Molecular Sciences, Computer Science, Mathematics, Geology, Seismology, Mechanical and Aerospace Engineering, Physics/Astrophysics.

This work, also describes the use of OpenMP (Open Multi-Processing) in multi-thread image processing applications. OpenMP is an extensive and powerful application-programming interface (API), supporting much functionality required for parallel programming [10]. The purpose of this work is to provide a high level image processing operation to demonstrate the ease of implementation and effectiveness of OpenMP in the image processing.

II. DIGITAL CLASSIC AND OPTIMIZED 2D FILTERS

A. Mean (average) filter classic and optimized

The arithmetic classic mean filter is defined as the average of all pixels spectrum within a local region of an image.

Fig. 1. Classic average filter.

Pseudocode of mean (average) classic filter (independent to kernel size)

// A original image in gray value,
// B processed image and K kernel matrix
// n, m image dimension
// kr  kernel rang ( kernel dimension = kr*2+1)
// ksize  kernel size
// xi, yi index pixel of image
// kx ky virtual element range [-kr,kr]

for yi=0 to m
   // correct processing loops
   for xi=0 to n
      // 1.- Take pixels from gray value image A
      // in kernel area and add to sum
      sum=0
      for ky=kr- to kr
         tyi = yi+ky
         if tyi<0 then tyi=0
         if tyi>= m then tyi=m-1
         for kx=kr- to kr
            sum=sum+A[tyi,kx]
            // Get pixels from image A and add to sum

   end
   end
   // 2.- Evaluate average from kernel matrix size
   prom=sum/( ksize)
   // 3.- Take average value and put in study pixel in image B
   B[yi,xi]=prom
   end

Fig. 2. Optimized average filter.

Pseudocode of mean (average) optimized filter (independent to kernel size)

// A original image in gray value,
// B processed image and K kernel matrix
// n, m image dimension
// kr  kernel rang ( kernel dimension = kr*2+1)
// ksize  kernel size
// xi, yi index pixel of image
// kx ky virtual element range [-kr,kr]
// txi tyi temporal index values to correct indexes which is out of image area

for yi=0 to m
   // correct processing loops
   for xi=0 to n
      // 1.- Take pixels from gray value image A
      // in kernel area and add to sum
      sum=0
      for ky=kr- to kr
         tyi = yi+ky
         if tyi<0 then tyi=0
         if tyi>= m then tyi=m-1
         for kx=kr- to kr
            if txi<0 then txi=0
            if txi>= m then txi=m-1
            sum=sum+A[tyi,txi]
            // Get pixels from image A and add to sum

end
end
// 2.- Evaluate average from kernel matrix size
prom=sum/( ksize)
// 3.- Take average value and put in study pixel in image B
B[yi,xi]=prom
end

The optimized mean filter obtained by accumulation of the neighborhood of pixel \( P(y,x) \), shares a lot of pixels in common with the accumulation for pixel \( P(y,x+1) \). This means that there is no need to compute the whole kernel for all pixels except only the first pixel in each row. Successive pixel filter response values can be obtained with just an add and a subtract to the previous pixel filter response value [11-14].
// 1.- Take pixels from gray value image A
// (in kernel area) and add to sum
// only for first pixel in the row
xi=0
for ky=kr- to kr
  tyi = yi+ky
  if tyi<0 then tyi=0
  if tyi>= m then tyi=m-1
  for kx=-kr to kr
    txi=xi+kx
    if txi<0 then txi=0
    if txi>= m then txi=m-1
    sum=sum+A[tyi,txi]  // Get pixels from image A
      // and add to sum
  end
end
// 2.- Evaluate average from kernel matrix
// size
prom=sum/( ksize)
// 3.- Take average value and put in study
// pixel in image B
B[yi,xi]=prom
for xi=1 to n
  // 4.- recursive recalculation of sum
  for ky=-kr to kr
    tyi = yi+ky
    if tyi<0 then tyi=0
    if tyi>= m then tyi=m-1
    ky=xi-kr-1
    if txi<0 then txi=0
    if txi>= m then txi=n-1
    // Subtract from the sum of the value of the pixel
    // from image A which is out of the kernel area
    sum=sum-A[tyi,txi]
    ky=xi+kr
    if txi<0 then txi=0
    if txi>= m then txi=n-1
    // Add to sum of the value of the pixel from image
    // A which is come into of the kernel area
    sum=sum+A[tyi,txi]
  end
// 5.- Evaluate average from kernel matrix size
prom=sum/(ksize)
// 6.- Take average value and put in study
// pixel in image B
B[yi,xi]=prom
end
end

B. Median filter classic and optimized

Classic median filter replaces the value of a pixel spectrum by the median of the spectrum levels in the neighborhood of that pixel.

Pseudocode of median classic filter
// A original image in gray value,
// B processed image and K kernel matrix
// n, m image dimension

// kd  kernel dimension
// xi, yi index pixel of image
// n, m image dimension
// kr kernel rang (kernel dimension = kr*2+1)
// ksize kernel size (kr*2+1)*(kr*2+1)
// kx ky virtual element range [-kr,kr]
// txi tyi temporal index values to correct
// indexes which is out of image area
// imed index of median value
// ksize = (kr*2+1)*(kr*2+1)
// imed = (ksize-1)/2
for yi=0 to m
  for xi=0 to n
    // 1.- Take pixels from gray value image A to array 1D
    ind=0
    for ky=-kr to kr
      tyi = yi+ky
      if tyi<0 then tyi=0
      if tyi>= m then tyi=m-1
      ky=xi-kr-1
      if txi<0 then txi=0
      if txi>= m then txi=n-1
      // Get pixels from image A and add to array
      vec[ind] =A[tyi,txi]
    end
    // 2. sort array 1D
    Sort(vec)
    // 3.- Take middle term from 1D array
    // and put in study pixel in image B
    vec[ind] =A[tyi,txi]
    B[yi,xi]=vec[imed]  // put processed pixel in
                      // processed image B
  end
end

Fig. 3. Classic median filter.

Median filtering is a commonly applied non-linear filtering technique that is particularly useful in removing speckle and
salt and pepper noise. It works by moving through the image pixel by pixel, and replacing each value with the median value of neighbouring pixels.

The optimized median filter is obtained through the histogram of spectrum for median calculation can be far more efficient because it is simple to update the histogram from window to window. Thus the histogram used for accumulating pixels in the kernel and only a part of it is modified when moving from one pixel to another [8-13], [16].

Fig. 4. Optimized median filter.

Pseudocode of median optimized filter

// A original image in gray value,
// B processed image and K kernel matrix
// n, m image dimension
// kd kernel dimension
// xi, yi index pixel of image
// n, m image dimension
// kr kernel rang ( kernel dimension = kr*2+1)
// ksize kernel size (kr*2+1)*(kr*2+1)
// kx ky virtual element range [-kr,kr]
// txi tyi temporal index values to correct indexes
// which is out of image area
// imed index of median value
// Hist histogram of intensity [0..255]
// medV value of median
// delta
ksize = (kr*2+1)*(kr*2+1)
for yi=0 to m
// 1. Clear histogram and fill it using kernel area values
  Clear(Hist) // only for first pixel in the row
  // can be conducted as
For i=0; to 256
  Hist[i]=0;
  xi=0;
  for ky=-kr to kr
    tyi = yi+ky
    if tyi<0 then tyi=0
    if tyi>m then tyi=m-1
    txi=xi-kr-1
    // Remove element from histogram
    if txi<0 then txi=0
    if txi>m then txi=m-1
    // Get pixels from image A to sValue
    sValue=A[tyi,txi]
    Hist[sValue]++ // increase count in histogram
  end
end
// 2. Find median index in histogram
medV = histogram_median(Hist, delta);
// 3. put median value in study pixel in image B
B[xi,yi]= medV
// 4. Recursively change histogram
for xi=1 to n
  for ky=-k to kr
    tyi = yi+ky
    if tyi<0 then tyi=0
    if tyi>m then tyi=m-1
    txi=xi-kr-1
    // Remove element from histogram
    if txi<0 then txi=0
    if txi>m then txi=m-1
    // Get pixels from image A to sValue
    sValue=A[tyi,txi]
    Hist[sValue]-- // decrease count in histogram
      // in index=sValue
    if sValue < medV then delta= delta-1
    if sValue > medV then delta= delta+1
    // Add element to histogram
    txi=xi+kr
    if txi<0 then txi=0
    if txi>m then txi=m-1
    // Get pixels from image A to sValue
    sValue=A[tyi,txi]
    Hist[sValue]++ // increase count in histogram
      // in index=sValue
    if sValue < medV then delta= delta-1
    if sValue > medV then delta= delta+1
end
// 5. Recalculate median index in histogram
medV = recalculate_histogram_median(Hist, delta);
// 6. Put median value in study pixel in image B
B[xi,yi]= medV
end
end

// Find median index in histogram
histogram_median(Hist,delta);
MCount=(ksize-1)/2
Res=0
Lcount =0;
for ind=0 to 255
  Lpcount= Lpcount+ Hist[ind]
  if Lcount < MCount then
    continue to next iteration;
  Else
    Res=ind
    break;
  end
delta= ksize- Hist[res]
return res
// Recalculate median index in histogram
recalculate_histogram_median(Hist, delta);
MCount=(ksize-1)/2

(Advance online publication: 20 November 2019)
Tmp_delta= delta
Tmp_med= medV
If Tmp_delta> MCount then
  While (Tmp_delta> MCount and Tmp_med>0)
    Tmp_med= Tmp_med-1
    If Hist[Tmp_med] >0 then  Tmp_delta= Tmp_delta-
    Hist[Tmp_med]
  Else
    While (Tmp_delta+ Hist[Tmp_med] <  MCount and
      Tmp_med<255)
      If Hist[Tmp_med] >0 then Tmp_delta= Tmp_delta+
      Hist[Tmp_med]
    Tmp_med= Tmp_med+1
  delta = Tmp_delta
return Tmp_med

C. Gauss filter 2D and optimized 1Dx2
The Gaussian 2D filter uses a Gaussian function (which also expresses the normal distribution in statistics) for calculating the transformation to apply to each pixel in the image.

![Fig. 5. Classic Gaussian 2D filter.](image)

Pseudocode of classic Gaussian 2D filter:
// 1.- Calculate kernel Gauss bell G(x,y).
GaussianCoef2D(RH,RW, sigma);
Sum=0;
for y= -RH to RH
  for x= -RW to RW
    Gxy[x+kr,y+kr]=(1/(2*pi*sigma*sigma))*exp(-
    (x*x+y*y)/(2*sigma*sigma))
    Sum=Sum+Gxy[x+kr,y+kr]
  end
end
Gxy=GaussianCoef2D(kr, kr,sigma);
for yi=0 to m    // correct processing loops
  for xi=0 to n
    // 2.- Take pixels from gray value image A
    // in kernel area and add to sum considering
    // Gaussian coefficient
    sum=0
    for ky=kr- to kr
      tyi = yi+ky
      if tyi<0 then tyi=0
      if tyi>= m then tyi=m-1
      for kx=kr- to kr
        txi=xi+kx
        if txi<0 then txi=0
        if txi>= m then txi=m-1
        // Get pixels from image A and multiply
        // it on Gaussian coefficient
        sum=sum+A[tyi,txi]*Gxy[ky+kr][kx+kr]
      end
    end
    // 3.- put obtained value in study pixel in image B
    B[yi,xi]=prom
  end
end

The convolution of Gaussian filter can be performed much faster since the equation for the 2D isotropic Gaussian is separable into y and x components [7-13], [15].

GaussianCoef1D(rang, sigma)
Sum=0
for t= -rang to rang
  G[t+rang]=(1/(sqrt(2*pi)*sigma))*exp((-t*t)/(2*sigma
  *sigma))
  Sum= Sum + G[t+ rang]
end
for t= -rang to rang
  G[t+ rang]= G[t+ rang]/ Sum
end

![Fig. 6. Optimized Gaussian 2D filter.](image)

Pseudocode of Gaussian 2D filter in double 1D interpretation:
// TMP_Image is transposed version of the image
// 1.- Calculate kernel Gauss in 1D.
Coef1D_1=GaussianCoef1D(kr,sigma);
Coef1D_2=GaussianCoef1D(kr,sigma);
/// Process 1D Gaussian (first)
for yi=0 to m // correct processing loops
    for xi=0 to n
        // 2.- Take pixels from gray value image A
        // in kernel area and add to sum considering
        // Gaussian coefficient
        sum=0;
        for kx=-r to r
            txi=xi+kx
            if txi<0 then txi=0
            if txi>= m then txi=n-1
        // Get pixels from image A and multiply it on
        // 1D Gaussian coefficient
        sum=sum+A[yi,txi]*Coef1D_1[[kx+kr]
        end
    // 3.- put obtained value in study pixel in
    // temporal image TMP_Image
    TMP_Image [xi,yi]= sum
/// Process 1D Gaussian (second)
for yi=0 to n // correct processing loops
    for xi=0 to m
        // 4.- Take pixels from gray value image TMP_Image
        // in kernel area and add to sum considering
        // Gaussian coefficient
        sum=0;
        for kx=-kr to kr
            txi=xi+kx
            if txi<0 then txi=0
            if txi>= m then txi=n-1
        // Get pixels from image A and multiply it on
        // 1D Gaussian coefficient
        sum=sum+TMP_Image [yi,txi]*Coef1D_2[[kx+kr]
        end
    // 5.- put obtained value in study pixel in image B
    B[xi,yi]= sum
end

III. OPENMP

Parallel Programming may speed up code. Today computers have one or more CPUs that have multiple processing cores (Multi-core processor). This helps with desktop computing tasks like multitasking (running multiple programs, plus the operating system, simultaneously). For scientific computing, this means the ability in principle of splitting up computations into groups and running each group on its own processor [18].

Two main paradigms talk about here are shared memory versus distributed memory models. In shared memory models, all multiple processing units have access to the same memory space. This is the case on desktop or laptop with multiple CPU cores. In a distributed memory model, multiple processing units each of their have their own memory store, and information is passed between them. This is the model that a networked cluster of computers operates with. A computer cluster is a collection of standalone computers that are connected to each other over a network, and are used together as a single system.

The methodology in our case of the algorithms (filters) for processing images is:
1.- Select Kernel
2.- Evaluate denoise filter with parallel OpenMP
   #pragma omp parallel for
   for (int y=0; y< Image_Height; y++)
       for (int x=0; x< Image_Width; x++)
           {
               // do denoise filters
           }
3.- Processed pixel put in study pixel of image denoise

OpenMP is an API that implements a multi-threaded, shared memory form of parallelism. It uses a set of compiler directives that are incorporated at compile-time to generate a multi-threaded version of program code. OpenMP is designed for multi-processor/core, shared memory machines [17], [18].

IV. METRICS: PSNR, SSIM

Any processing applied to an image may cause an important loss of information or quality. Image quality evaluation methods can be subdivided into objective and subjective methods. Subjective methods are based on human judgment and operate without reference to explicit criteria. Objective methods are based on comparisons using explicit numerical criteria, and several references are possible such as the ground truth or prior knowledge expressed in terms of statistical parameters and tests.

The next equations show the relationship between the SSIM (structural similarity index measure) and the PSNR (peak-signal-to-noise ratio) for grey-level (8 bits) images. Given a reference image f and a test image g, both of size M×N, the PSNR between f and g is defined by:

\[
PSNR(f, g) = 10 \log_{10} \left( \frac{255^2}{MSE(f, g)} \right)
\]

where

\[
MSE(f, g) = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} (f_{ij} - g_{ij})^2
\]

The PSNR value approaches infinity as the MSE approaches zero; this shows that a higher PSNR value provides a higher image quality. At the other end of the scale, a small value of the PSNR implies high numerical differences between images. The SSIM is a quality metric used to measure the similarity between two images. Wang et al. developed it, and it is correlated with the quality perception of the human visual system (HVS). Instead of using traditional error summation methods, the SSIM is designed by modeling any image distortion as a combination of three factors that are loss of correlation, luminance distortion and contrast distortion. The SSIM is defined as:

\[
SSIM(f, g) = l(f, g)c(f, g)s(f, g).
\]
where:

\[
\begin{align*}
l(f, g) &= \frac{2\mu_f \mu_g + C_1}{\mu_f^2 + \mu_g^2 + C_1} \\
c(f, g) &= \frac{2\sigma_f \sigma_g + C_2}{\sigma_f^2 + \sigma_g^2 + C_2} \\
s(f, g) &= \frac{\sigma_{fg} + C_3}{\sigma_f \sigma_g + C_3}
\end{align*}
\]

The first term is the luminance comparison, function that measures the closeness of the two images’ means luminance ($\mu_f$ and $\mu_g$). This factor is maximal and equal to 1 only if $\mu_f = \mu_g$. The second term is the contrast comparison, function that measures the closeness of the contrast of the two images. Here the contrast is measured by the standard deviation $\sigma_f$ and $\sigma_g$. This term is maximal and equal to 1 only if $\sigma_f = \sigma_g$.

The third term is the structure comparison, function that measures the correlation coefficient between the two images $f$ and $g$.

Note that $\sigma_{fg}$ is the covariance between $f$ and $g$. The positive values of the SSIM index are in $[0,1]$. A value of 0 means no correlation between images, and 1 means that $f=g$.

The positive constants $C_1$, $C_2$ and $C_3$ are used to avoid a null denominator [21].

V. EXPERIMENTAL RESULTS

Different X-ray images, with different sizes were processed with the classic and optimized filters: mean, median, Gaussian 2D.

The experiment used a PC based on Intel Core i5 3.1 GHz with 8 GB RAM. The results were obtained by measuring the processing time of 80 different images (for each image, 400 measurements were taken).

Also, the acceleration stability of optimized algorithms was evaluated depending on the size of the core and the number of threads used. To estimate the acceleration, the mean values obtained during the 300 measurements were taken for each combination of the kernel size and the number of threads. Fig. 10 shows the average values of the obtained acceleration coefficient for optimized versions of filters.

The experimental results show that the increase in the processing speed for different kernel sizes is almost the same. Some stability is observed in the acceleration for two threads as well as one can see the increase of the acceleration coefficient in the case with more than two threads having the kernel size larger than 5x5. Acceleration with the usage of four threads demonstrates poor efficiency as parts of the CPU resources are spent on background tasks (Fig. 10).

Fig. 7. Shared memory in OpenMP.

Fig. 8 show processing time of filters classic and optimized for different image size (4 threads OpenMP). In addition, a study of optimized filter implementations was made. It showed the magnitude of the acceleration relative to the sequential implementation of the classical version of the filters. The result of this study was represented as the maps of acceleration of optimized filters, showing acceleration coefficient depending on the kernel size (Fig. 9). The maps were generated as average values obtained for different image sizes.

Also, the acceleration stability of optimized algorithms was evaluated depending on the size of the core and the number of threads used. To estimate the acceleration, the mean values obtained during the 300 measurements were taken for each combination of the kernel size and the number of threads. Fig. 10 shows the average values of the obtained acceleration coefficient for optimized versions of filters.

The experimental results show that the increase in the processing speed for different kernel sizes is almost the same. Some stability is observed in the acceleration for two threads as well as one can see the increase of the acceleration coefficient in the case with more than two threads having the kernel size larger than 5x5. Acceleration with the usage of four threads demonstrates poor efficiency as parts of the CPU resources are spent on background tasks (Fig. 10).

![Processing time of classic implementation of filters in parallel mode with 4 threads (Kernel 5x5)](image)

![Processing time of optimized filters in parallel mode with 4 threads (Kernel 5x5)](image)

<table>
<thead>
<tr>
<th>Kernel width</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>9</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel height</td>
<td>3</td>
<td>1.41</td>
<td>1.97</td>
<td>2.53</td>
<td>3.12</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.56</td>
<td>2.16</td>
<td>2.93</td>
<td>3.63</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>1.57</td>
<td>2.35</td>
<td>3.11</td>
<td>3.84</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>1.60</td>
<td>2.39</td>
<td>3.17</td>
<td>3.93</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>1.61</td>
<td>2.42</td>
<td>3.24</td>
<td>4.06</td>
</tr>
</tbody>
</table>

Fig. 8. Processing time of filters (ms.) for different image size using 4 threads OpenMP: (a) classic implementation; (b) optimized implementation

(Advance online publication: 20 November 2019)
To assess the overall acceleration, maps were constructed showing the acceleration values of the parallel implementation of optimized algorithms (4 threads) relative to the classical implementation (Fig. 11) for different kernel sizes.

In addition, evaluations of noise suppression characteristics were also performed. To simulate the noise, which may occur in the equipment, were put layered noise over the image, the additive noise part was 80%, and while the impulse noise part was 20%.

A. Processing Medical X-Ray Images

This part shows the results of the suppressing noise of three X-Ray images of different anatomical structure and with principal aim to allow the medical expert give us their opinion about the quality and the utility for diagnostic.

Basically for each image, 300 noise maps were generated, which were superimposed on the original image. After that, filters (with different kernel sizes) were applied to the noisy image and the PSNR and SSIM metrics were calculated.

Pelvis X-Ray image:

The figure 12 presents the images of a Pelvis, (a) is the original image obtained from the equipment; (b), (c), (d) show the image processing through algorithms: Gaussian filter, Mean filter and Median filter respectively; (e) is the image added noise and (f) is the original image again.
Fig. 12. Processing of Pelvis image: (a) original image obtained of equipment; (b) Image processing using the Gaussian filter algorithm; (c) Image processing using the Mean filter algorithm; (d) Image processing using the Median filter algorithm; (e) Original image added noise over 20%; (f) Original image again. All processing were done using the OpenMP.

Table I demonstrates average PSNR values in dB for optimized filters when processing noise images with size 4280×3520 pixels. Table II demonstrates average SSIM values (multiplied by 100).

### Table I
**PSNR VALUES (dB) OF OPTIMIZED FILTERS.**

<table>
<thead>
<tr>
<th>Noise data Level</th>
<th>Kernel Size</th>
<th>PSNR</th>
<th>Filter Size</th>
<th>Mean</th>
<th>Gauss</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>3x3</td>
<td>29.4861</td>
<td>29.4266</td>
<td>48.3976</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5x5</td>
<td>33.4550</td>
<td>32.9562</td>
<td>48.7454</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15%</td>
<td>7x7</td>
<td>35.4710</td>
<td>35.7210</td>
<td>46.2457</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11x11</td>
<td>36.8100</td>
<td>36.9227</td>
<td>47.4257</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3x3</td>
<td>27.8072</td>
<td>27.8463</td>
<td>46.9735</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5x5</td>
<td>32.8989</td>
<td>32.9562</td>
<td>47.1794</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20%</td>
<td>7x7</td>
<td>36.8100</td>
<td>36.9227</td>
<td>47.4257</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9x9</td>
<td>37.1053</td>
<td>37.0847</td>
<td>48.2768</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11x11</td>
<td>42.2185</td>
<td>42.2055</td>
<td>49.2208</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5x5</td>
<td>69.6078</td>
<td>67.5101</td>
<td>97.1962</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25%</td>
<td>7x7</td>
<td>42.2185</td>
<td>42.2055</td>
<td>49.2208</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9x9</td>
<td>87.4868</td>
<td>79.6765</td>
<td>96.5805</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11x11</td>
<td>90.7826</td>
<td>80.0833</td>
<td>96.4585</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table II
**SSIM PERCENT VALUES OF OPTIMIZED FILTERS.**

<table>
<thead>
<tr>
<th>Noise data Level</th>
<th>Kernel Size</th>
<th>SSIM</th>
<th>Filter Size</th>
<th>Mean</th>
<th>Gauss</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>3x3</td>
<td>18.8959</td>
<td>58.9344</td>
<td>58.9526</td>
<td>98.1160</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5x5</td>
<td>80.4354</td>
<td>78.9749</td>
<td>97.2779</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9x9</td>
<td>92.2250</td>
<td>86.5102</td>
<td>96.6490</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11x11</td>
<td>93.6385</td>
<td>86.7419</td>
<td>96.5242</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3x3</td>
<td>49.1308</td>
<td>49.1249</td>
<td>97.8409</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5x5</td>
<td>74.5419</td>
<td>73.6822</td>
<td>97.2388</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15%</td>
<td>7x7</td>
<td>12.3248</td>
<td>85.8297</td>
<td>84.5850</td>
<td>96.8320</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9x9</td>
<td>90.5334</td>
<td>86.3964</td>
<td>96.6290</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11x11</td>
<td>92.6264</td>
<td>89.7023</td>
<td>96.5052</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3x3</td>
<td>42.2185</td>
<td>42.2055</td>
<td>97.2208</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5x5</td>
<td>69.6078</td>
<td>67.5101</td>
<td>97.1962</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20%</td>
<td>7x7</td>
<td>9.2232</td>
<td>83.0469</td>
<td>82.6568</td>
<td>96.8051</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9x9</td>
<td>88.9999</td>
<td>84.8045</td>
<td>96.6074</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11x11</td>
<td>91.6793</td>
<td>87.1676</td>
<td>96.4844</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3x3</td>
<td>37.1053</td>
<td>37.0847</td>
<td>96.1048</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5x5</td>
<td>65.4875</td>
<td>64.2323</td>
<td>97.1492</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25%</td>
<td>7x7</td>
<td>6.4762</td>
<td>80.5558</td>
<td>78.2772</td>
<td>96.7751</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9x9</td>
<td>87.4868</td>
<td>79.6765</td>
<td>96.5805</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11x11</td>
<td>90.7826</td>
<td>80.0833</td>
<td>96.4585</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Tibia Fracture X-Ray image:**

The figure 13 shows the images of a Tibia fracture, (a) is the original image obtained from the equipment; (b), (c), (d) show the image processing through algorithms: Gaussian filter, Mean filter and Median filter respectively; (e) is the image added noise and (f) is the original image again.
Table III demonstrates average PSNR values in dB for optimized filters when processing noise image of the fracture with size 3157×3831 pixels. Table IV demonstrates average SSIM values (multiplied by 100).

Chest and lung X-Ray image:

The figure 14 shows the images of a Chest and lung. (a) is the original image obtained from the equipment; (b), (c), (d) show the image processing through algorithms: Gaussian filter, Mean filter and Median filter respectively; (e) is the image added noise and (f) is the original image again.

Table V demonstrates average PSNR values in dB for optimized filters when processing noise image of the chest with size 3944×3205 pixels. Table VI demonstrates average SSIM values (multiplied by 100).
Brief analysis of medical images processing

In order to validate the results of the experiments, after the processing of the images, it was requested to ten medical experts to give their point of view about the quality of the images. Experts were professionals of different medical specializations (oncology, traumatology and surgery).

The table VII presents the percentage of the selection of the image accord the enumeration with figure 12, 13 and 14.

<table>
<thead>
<tr>
<th>Noise data Level</th>
<th>Kernel Size</th>
<th>PSNR Mean</th>
<th>PSNR Gauss</th>
<th>PSNR Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>10% 20.0350</td>
<td>3x3</td>
<td>29.4543</td>
<td>29.3957</td>
<td>50.4992</td>
</tr>
<tr>
<td></td>
<td>5x5</td>
<td>33.5793</td>
<td>33.0090</td>
<td>47.8523</td>
</tr>
<tr>
<td>15% 18.3514</td>
<td>7x7</td>
<td>35.9817</td>
<td>34.4005</td>
<td>46.2378</td>
</tr>
<tr>
<td></td>
<td>9x9</td>
<td>37.4995</td>
<td>34.8000</td>
<td>45.4176</td>
</tr>
<tr>
<td></td>
<td>11x11</td>
<td>38.4829</td>
<td>34.8804</td>
<td>44.8985</td>
</tr>
<tr>
<td></td>
<td>3x3</td>
<td>27.7304</td>
<td>27.6728</td>
<td>48.6415</td>
</tr>
<tr>
<td></td>
<td>5x5</td>
<td>31.8051</td>
<td>31.2383</td>
<td>47.7588</td>
</tr>
<tr>
<td>20% 17.0729</td>
<td>7x7</td>
<td>32.7961</td>
<td>31.2705</td>
<td>46.1344</td>
</tr>
<tr>
<td></td>
<td>9x9</td>
<td>35.1099</td>
<td>34.6748</td>
<td>45.3676</td>
</tr>
<tr>
<td></td>
<td>11x11</td>
<td>35.1740</td>
<td>34.7236</td>
<td>44.8635</td>
</tr>
<tr>
<td></td>
<td>3x3</td>
<td>29.5474</td>
<td>28.9419</td>
<td>42.6366</td>
</tr>
<tr>
<td></td>
<td>5x5</td>
<td>30.4996</td>
<td>29.9449</td>
<td>41.6548</td>
</tr>
<tr>
<td>25% 16.2156</td>
<td>7x7</td>
<td>31.6746</td>
<td>30.2088</td>
<td>46.1113</td>
</tr>
<tr>
<td></td>
<td>9x9</td>
<td>33.0452</td>
<td>30.5753</td>
<td>45.3394</td>
</tr>
<tr>
<td></td>
<td>11x11</td>
<td>33.9270</td>
<td>30.6481</td>
<td>44.8421</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Noise data Level</th>
<th>Kernel Size</th>
<th>SSIM Mean</th>
<th>SSIM Gauss</th>
<th>SSIM Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>10% 20.5374</td>
<td>3x3</td>
<td>59.1579</td>
<td>59.1921</td>
<td>97.5027</td>
</tr>
<tr>
<td></td>
<td>5x5</td>
<td>79.7355</td>
<td>78.4048</td>
<td>96.4102</td>
</tr>
<tr>
<td></td>
<td>7x7</td>
<td>88.1099</td>
<td>85.4020</td>
<td>95.9976</td>
</tr>
<tr>
<td></td>
<td>9x9</td>
<td>91.4174</td>
<td>87.7512</td>
<td>95.7559</td>
</tr>
<tr>
<td></td>
<td>11x11</td>
<td>92.8591</td>
<td>88.9772</td>
<td>95.6168</td>
</tr>
<tr>
<td></td>
<td>3x3</td>
<td>49.2356</td>
<td>49.2464</td>
<td>97.2201</td>
</tr>
<tr>
<td></td>
<td>5x5</td>
<td>73.7181</td>
<td>72.9807</td>
<td>96.3685</td>
</tr>
<tr>
<td>15% 12.9862</td>
<td>7x7</td>
<td>84.8214</td>
<td>80.6935</td>
<td>95.9556</td>
</tr>
<tr>
<td></td>
<td>9x9</td>
<td>89.4914</td>
<td>82.4693</td>
<td>97.5400</td>
</tr>
<tr>
<td></td>
<td>11x11</td>
<td>91.5983</td>
<td>84.6796</td>
<td>95.6063</td>
</tr>
<tr>
<td></td>
<td>3x3</td>
<td>42.0494</td>
<td>42.0947</td>
<td>96.0204</td>
</tr>
<tr>
<td></td>
<td>5x5</td>
<td>68.6288</td>
<td>66.3334</td>
<td>95.3285</td>
</tr>
<tr>
<td>20% 8.7338</td>
<td>7x7</td>
<td>81.5181</td>
<td>76.5684</td>
<td>95.9369</td>
</tr>
<tr>
<td></td>
<td>9x9</td>
<td>87.6384</td>
<td>80.6704</td>
<td>95.7251</td>
</tr>
<tr>
<td></td>
<td>11x11</td>
<td>90.3543</td>
<td>83.0265</td>
<td>95.5941</td>
</tr>
<tr>
<td></td>
<td>3x3</td>
<td>36.8910</td>
<td>36.8910</td>
<td>95.4997</td>
</tr>
<tr>
<td></td>
<td>5x5</td>
<td>64.4367</td>
<td>62.2856</td>
<td>96.2804</td>
</tr>
<tr>
<td>25% 6.4832</td>
<td>7x7</td>
<td>79.1728</td>
<td>74.0838</td>
<td>95.9190</td>
</tr>
<tr>
<td></td>
<td>9x9</td>
<td>85.9831</td>
<td>78.4279</td>
<td>95.7085</td>
</tr>
<tr>
<td></td>
<td>11x11</td>
<td>89.2582</td>
<td>81.0263</td>
<td>95.5819</td>
</tr>
</tbody>
</table>

V. CONCLUSIONS

Experiments were conducted to estimate the processing time of optimized filtering algorithms (Mean filter, Median filter, Gaussian filter) and evaluation of noise suppression.

Since medical point of view any radiological image presents an acceptable amount of noise, however, the important thing to considerate when reduce the noise, is that this amount of noise does not affect the quality of the image and especially the medical diagnosis.

The filters that were used in the study have been interpreted by different medical specialists, and depending on the pathology to be discarded or confirmed, they have validated the importance of image noise processing; it has also been possible to identify and have agreed that all radiographic images have noise, and its increase or decrease will be according to its diagnosis or interpretation in reference to identify soft tissue or bone tissue.

In order to the processing, the experimental results show that the increase in the processing speed for different kernel sizes is almost the same. Some stability is observed in the acceleration for two threads as well as one can see the increase of the acceleration coefficient in the case with more than two threads having the kernel size larger than 5x5. Acceleration with the usage of four threads demonstrates reduced efficiency as parts of the CPU resources are spent on background tasks.

Using OpenMP, we made parallel implementation of optimized algorithms, which gives performance boost up in almost two times for two threads and around 3, 2 times for 3 experts are distributed principally around the Original Image (1-3(a) and 1-3(f)) and the processing images using the Mean and Median filter algorithms (1-3(c), 1-3(d)).

Accord to the medical experts the “clinical eye” require training and depend also of the different diseases and tissues that seeking in the image.

In the daily medical practice by doctors of different specializations (orthopedist, traumatology, oncologists, neurologists), imaging studies cover approximately 80%, often becoming the first requested study, therefore an excessive amount of noise in a radiography is a limiting factor in the performance of the doctor, or interfere with the interpretation of the image.

Pelvis X-Ray image was processed by filters Mean, Gauss and Median, with kernel size 3x3, 5x5, 7x7, 9x9, 11x11 for level noise 10%, 15%, 20%, 25%, while the noise level increases PSNR decreases from 19.422 to 15.6150, and SSIM decreases from 18.8959 to 6.4762.

Tibia Fracture X-Ray image was processed by filters Mean, Gauss and Median, with kernel size 3x3, 5x5, 7x7, 9x9, 11x11 for level noise 10%, 15%, 20%, 25%, while the noise level increases PSNR decreases from 19.9981 to 16.3125, and SSIM decreases from 26.1078 to 8.0893.

Chest and lung X-Ray image was processed by filters Mean, Gauss and Median, with kernel size 3x3, 5x5, 7x7, 9x9, 11x11 for level noise 10%, 15%, 20%, 25%, while the noise level increases PSNR decreases from 20.0350 to 16.2156, and SSIM decreases from 20.5374 to 6.4832.

We can see that the percentages of the validation of the
and 4 threads. Experimental results demonstrate that the optimized version of filter algorithms can well do with the noise reduction at appropriate minimum processing time compared to classical implementation. The greatest increase of processing speed was gained for the median filter.

The experimental results also show that Median filter demonstrates the best noise reduction; though in some cases it suppresses details. Also, the Median filter requires the most computational time. For median filter optimal kernel size 3×3 and 5×5. Gaussian and mean filters give good results with kernel size 5×5 and larger depending of the image.

REFERENCES


