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

An 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 criterions 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

Manuscript received April 22, 2018; revised July 26, 2019. This work was supported by Universidad de las Fuerzas Armadas ESPE, Av. Gral Ruminahui s/n, Sangolqui Ecuador.

L. Cadena, is with Electric and Electronic Department at Universidad de las Fuerzas Armadas ESPE, Av. Gral Ruminahui s/n, Sangolqui Ecuador. (phone: +593997221212; e-mail: ecuadorx@gmail.com).

D. Castillo is with the Chemistry and Exact Sciences Department, Universidad Técnica Particular de Loja UTPL, Av. Marcelino Champagñat s/n, Loja Ecuador (e-mail: dpcastillo@utpl.edu.ec). Also D. Castillo are with Instituto de Instrumentación para Imagen Molecular (i3M) Universitat Politècnica de València – Consejo Superior de Investigaciones Científicas (CSIC), Camino de Vera s/n 46022 Valencia, Spain.

A. Zotin is with Department of Informatics and Computer Techniques, Reshetnev Siberian State University of Science and Technology, 31 krasnoyarsky rabochy av., Krasnoyarsk 660037, Russian Federation (e-mail: zotin@sibsau.ru).

F. Cadena is with College Unidad Educativa Atahualpa. Oyacoto-Quito, Ecuador. (e-mail: fcfc041@gmail.com)

P. Diaz, Health and Medical Science Department at Universidad Técnica Particular de Loja UTPL, Av. Marcelino Champagñat s/n, Loja Ecuador. (e-mail: pvdiaz@utpl.edu.ec)

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.

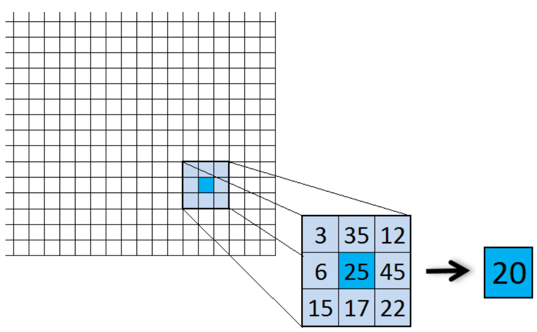


Fig. 1. Classic average filter.

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]
ksize=(kr*2+1)*(kr*2+1)
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 ky=kr- to kr
                if txi<0 then txi=0
                if txi>= m then txi=n-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
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].

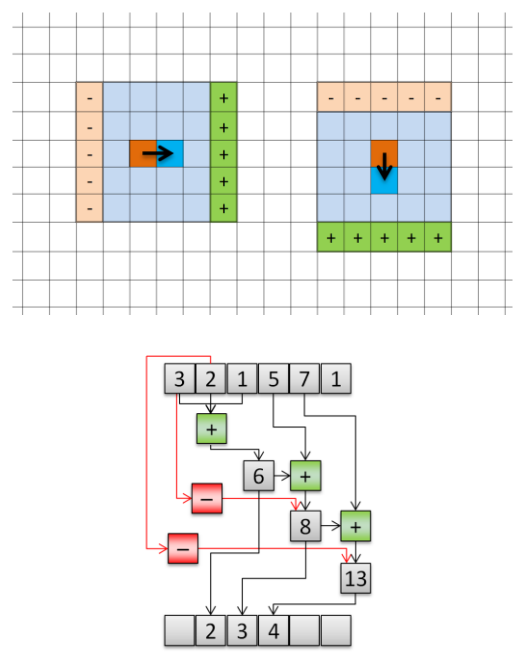


Fig. 2. Optimized average filter.

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
ksize=(kr*2+1)*(kr*2+1)
for yi=0 to m.
```

```
// 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 ky=kr- to kr
          if txi<0 then txi=0
          if txi>= m then txi=n-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 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
        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 add to array
           vec[ind] =A[tyi,txi]
        end
     end
        // 2. sort array 1D
        Sort(vec)
        // 3.- Take middle term from 1D array
        // and put in study pixel in image B
        B[xi,yi]=vec[imed] // put processed pixel in
                           // processed image B
     end
end
```

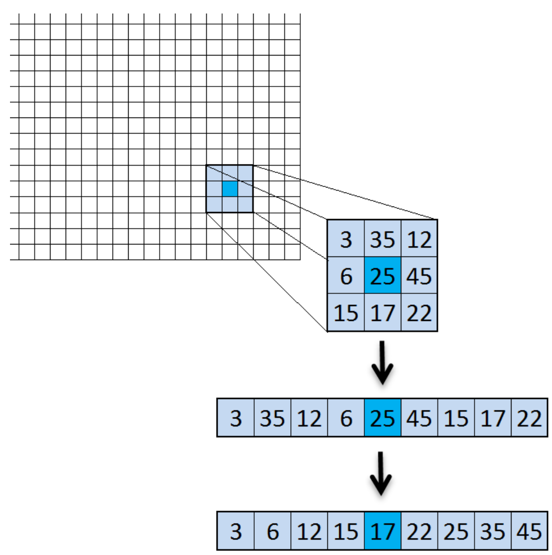


Fig. 3. Classic median filter.

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 trough 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].

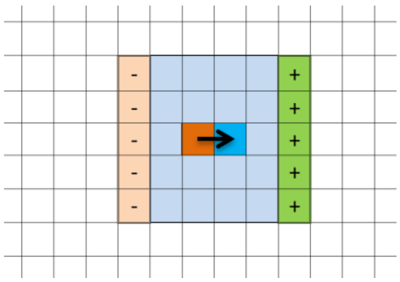


Fig. 4. Optimized median filter.

```
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
    for kx=-kr to kr
      if txi<0 then txi=0
      if txi>= m then txi=n-1
      // Get pixels from image A to sValue
      sValue =A[tyi,txi]
      Hist[sValue]++     // increase count in histogram
                         // in index=sValue
    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=n-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=n-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
```

```
  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.

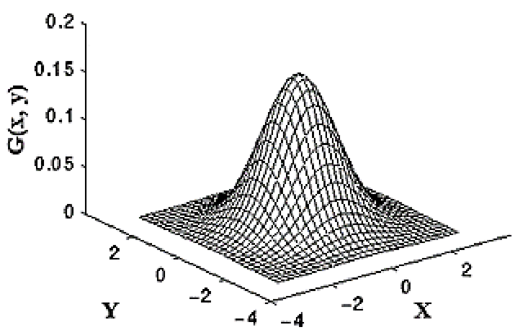


Fig. 5. Classic Gaussian 2D filter.

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]
for y= -RH to RH
  for x= -RW to RW
    Gxy[x+kr,y+kr] = Gxy[x+kr,y+kr]/Sum
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=n-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]
for t= -rang to rang
  G[t+ rang]= G[t+ rang]/ Sum
end
```

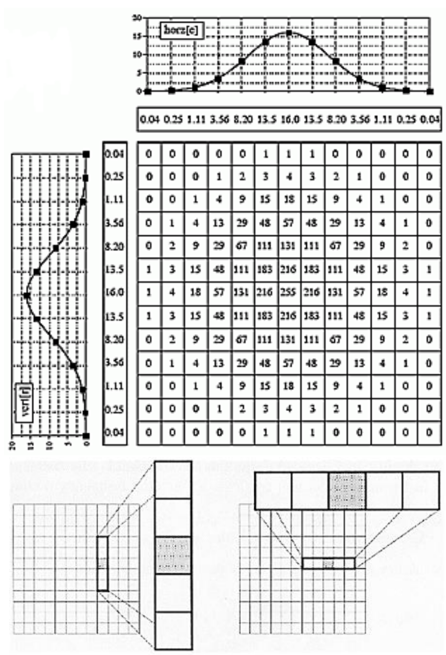


Fig. 6. Optimized Gaussian 2D filter.

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
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:

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

where

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

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:

$$l(f,g) = \frac{2\mu_f\mu_g + C_1}{\mu_f^2 + \mu_g^2 + C_1}$$
$$c(f,g) = \frac{2\sigma_f\sigma_g + C_2}{\sigma_f^2 + \sigma_g^2 + C_2}$$
$$s(f,g) = \frac{\sigma_{fg} + C_3}{\sigma_f\sigma_g + C_3}$$

The first term is the luminance comparison, function that measures the closeness of the two images' means luminance (μf and μg). 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 σf and σg. 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 σfg 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 C1, C2 and C3 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).

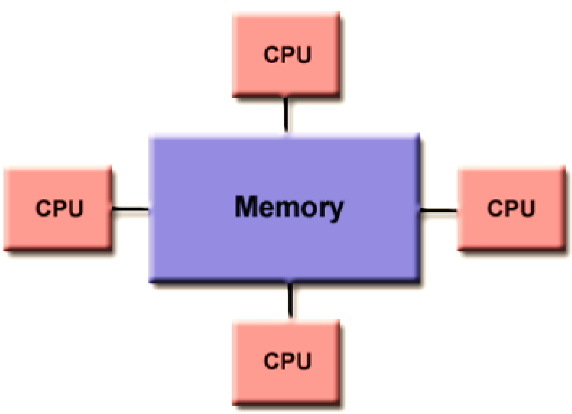


Fig. 7. Shared memory in OpenMP.

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 5×5. Acceleration with the usage of four threads demonstrates poor efficiency as parts of the CPU resources are spent on background tasks (Fig. 10).

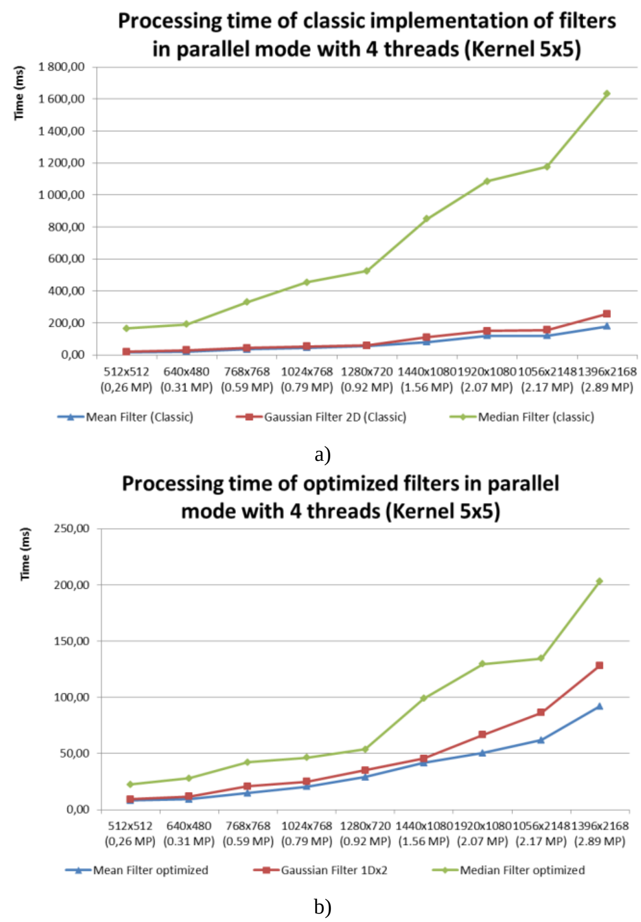


Fig. 8. Processing time of filters (ms.) for different image size using 4 threads OpenMP: (a) classic implementation; (b) optimized implementation

		Kernel width				
		3	5	7	9	11
Kernel heigth	3	1.41	1.97	2.53	3.12	3.68
	5	1.56	2.16	2.93	3.63	4.32
	7	1.57	2.35	3.11	3.84	4.56
	9	1.60	2.39	3.17	3.93	4.68
	11	1.61	2.42	3.24	4.06	4.89

a)

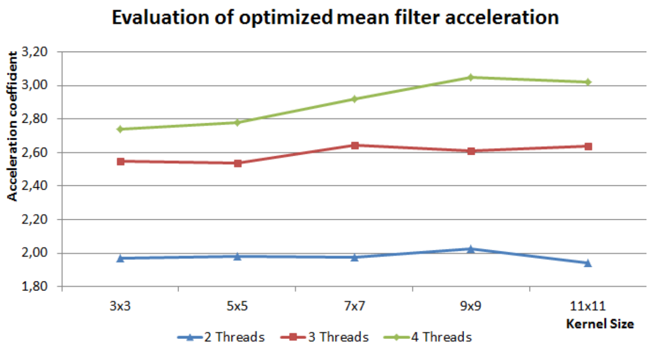
		Kernel width				
		3	5	7	9	11
Kernel heigth	3	1.14	1.46	1.62	1.82	1.94
	5	1.47	1.95	2.38	2.65	2.85
	7	1.70	2.35	2.88	3.28	3.46
	9	1.91	2.53	3.22	3.61	4.00
	11	1.95	2.74	3.45	4.14	4.72

b)

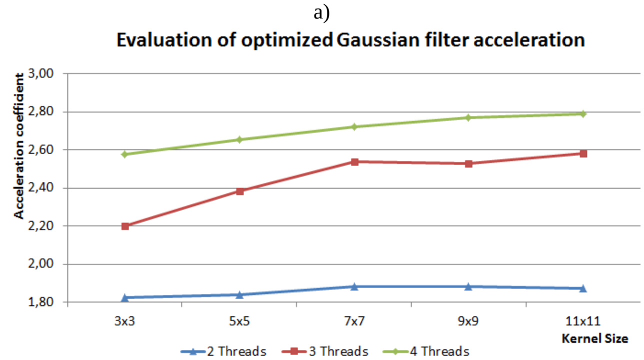
		Kernel width				
		3	5	7	9	11
Kernel heigth	3	4.40	7.60	11.21	14.71	19.22
	5	5.28	9.24	13.91	18.16	22.67
	7	5.88	10.62	15.29	20.09	25.93
	9	6.09	10.98	15.98	21.99	27.59
	11	6.66	11.37	17.48	23.39	29.11

c)

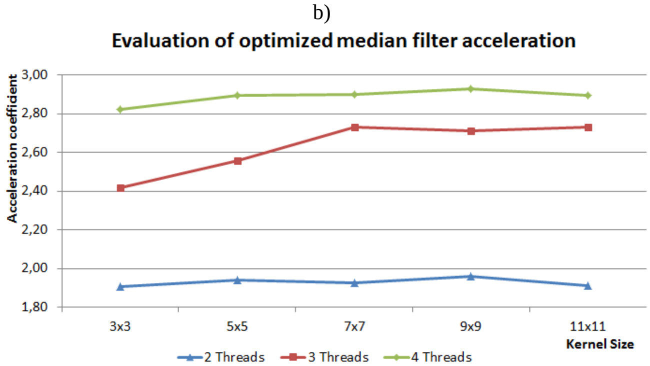
Fig. 9. Maps of acceleration of optimized filters compared to classical sequential implementation: (a) Mean filter; (b) Gaussian filter; (c) Median filter.



a)



b)



c)

Fig. 10. Evaluation of optimized filters acceleration: (a) Mean filter; (b) Gaussian filter; (c) Median filter.

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.

		Kernel width				
		3	5	7	9	11
Kernel heigth	3	4.80	5.90	7.98	8.49	10.76
	5	4.75	7.73	9.41	12.29	12.81
	7	5.21	7.65	10.23	12.57	13.78
	9	5.39	7.92	10.30	12.52	13.82
	11	5.95	6.85	9.28	11.55	14.29

a)

		Kernel width				
		3	5	7	9	11
Kernel heigth	3	3.58	4.43	5.15	5.12	5.46
	5	4.07	6.36	7.51	7.78	8.35
	7	5.14	6.53	8.29	9.11	9.36
	9	5.01	8.39	8.64	9.99	10.99
	11	5.55	8.41	9.51	10.45	12.23

b)

		Kernel width				
		3	5	7	9	11
Kernel heigth	3	13.34	24.92	35.77	44.89	54.31
	5	16.47	30.90	44.75	59.71	66.88
	7	18.54	30.94	47.84	61.42	71.95
	9	18.53	31.35	45.63	67.47	82.21
	11	21.47	35.78	49.40	65.77	84.29

c)

Fig. 11. Maps of acceleration of optimized filters (4 threads OpenMP) compared to classical sequential implementation: (a) Mean filter; (b) Gaussian filter; (c) Median filter.

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%.

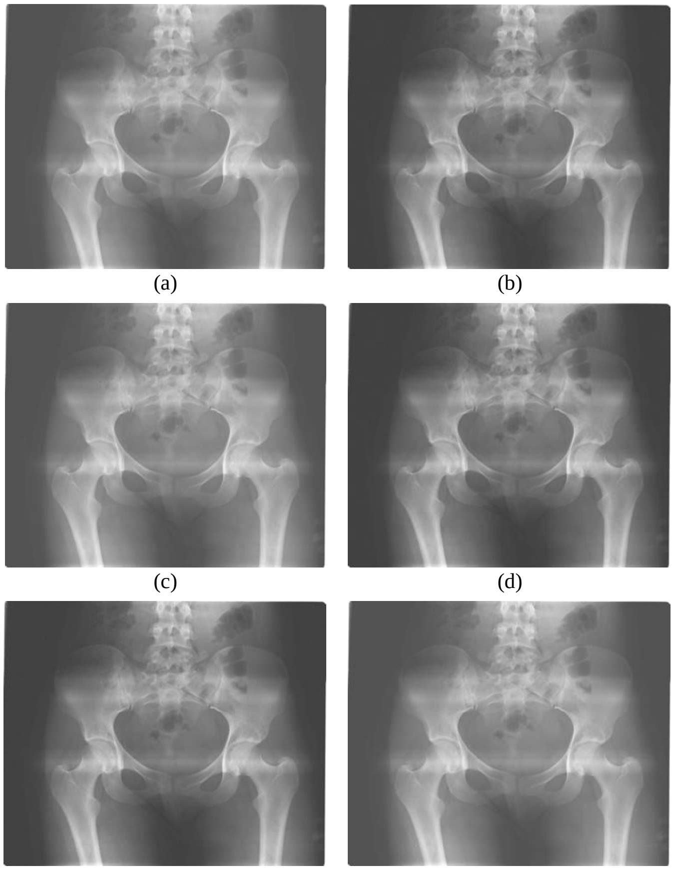
A. Processing Medical X-Ray Images

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.



(a) (b) (c) (d) (e) (f)

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 did it 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
PSRN VALUES (DB) OF OPTIMIZED FILTERS.

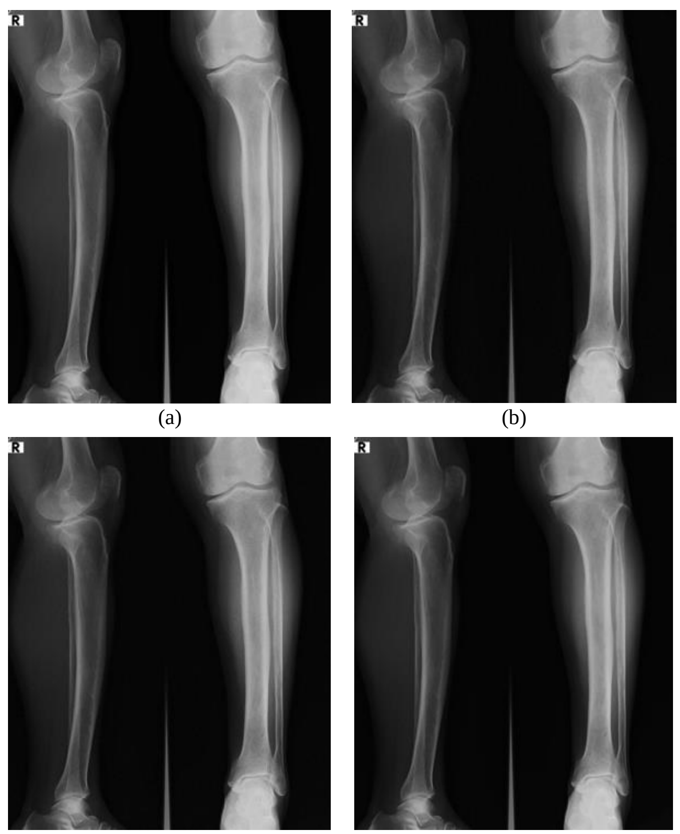
Noise data Level	PSNR	Kernel Size	Mean	Filter Gauss	Median
		3×3	29.4861	29.4266	48.3976
		5×5	33.4550	32.9562	48.7454
10%	19.4337	7×7	35.4710	34.8571	46.1494
		9×9	36.4492	35.7210	46.2457
		11×11	36.8100	35.6927	44.9220
		3×3	27.8072	27.8463	46.9735
		5×5	31.8449	31.6124	48.2768
15%	17.7498	7×7	34.0264	33.6532	45.9774
		9×9	35.2106	34.0320	45.9569
		11×11	35.7759	34.1073	44.7179
		3×3	26.6220	26.5610	44.6786
		5×5	30.6763	30.5295	47.8836
20%	16.4937	7×7	32.9232	32.4805	45.7714
		9×9	34.2034	33.8636	45.6254
		11×11	34.8803	34.0940	44.4509
		3×3	25.7089	25.6483	41.9746
		5×5	29.7512	29.5002	47.4195
25%	15.6150	7×7	32.0137	31.5455	45.5098
		9×9	33.3342	32.9273	45.2661
		11×11	34.0708	33.0036	44.1495

TABLE II
SSIM PERCENT VALUES OF OPTIMIZED FILTERS.

Noise data Level	SSIM	Kernel Size	Mean	Filter Gauss	Median
		3×3	58.9344	58.9526	98.1160
		5×5	80.4354	78.9749	97.2779
10%	18.8959	7×7	88.9184	85.1330	96.8562
		9×9	92.2250	86.5102	96.6490
		11×11	93.6385	86.7419	96.5242
		3×3	49.1308	49.1249	97.8409
		5×5	74.5419	73.6822	97.2388
15%	12.3248	7×7	85.8297	84.5850	96.8320
		9×9	90.5334	86.3964	96.6290
		11×11	92.6264	89.7023	96.5052
		3×3	42.2185	42.2055	97.2208
		5×5	69.6078	67.5101	97.1962
20%	9.2232	7×7	83.0469	82.6568	96.8051
		9×9	88.9499	84.8045	96.6074
		11×11	91.6793	87.1676	96.4844
		3×3	37.1053	37.0847	96.1048
		5×5	65.4875	64.2323	97.1492
25%	6.4762	7×7	80.5558	78.2772	96.7751
		9×9	87.4868	79.6765	96.5805
		11×11	90.7826	80.0833	96.4585

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.



(a) (b) (c) (d)

(e) (f)

Fig. 13. Processing of Tibia Fracture 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 did it using the OpenMP.

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).

TABLE III
PSRN VALUES (DB) OF OPTIMIZED FILTERS.

Noise data			Filter		
Level	PSNR	Kernel Size	Mean	Gauss	Median
		3×3	29.4861	29.4266	48.3976
		5×5	33.4550	32.9562	48.7454
10%	19.9981	7×7	35.4710	34.2571	46.1494
		9×9	36.4492	34.6210	46.2457
		11×11	36.8100	34.6927	44.9220
		3×3	27.8072	27.7463	46.9735
		5×5	31.8449	31.3124	48.2768
15%	18.4005	7×7	34.0264	32.6532	45.9774
		9×9	35.2106	33.0320	45.9569
		11×11	35.7759	33.1073	44.7179
		3×3	26.6220	26.5610	44.6786
		5×5	30.6763	30.1295	47.8836
20%	17.1854	7×7	32.9232	31.4805	45.7714
		9×9	34.2034	31.8636	45.6254
		11×11	34.8803	31.9400	44.4509
		3×3	25.7089	25.6483	41.9746
		5×5	29.7512	29.2002	47.4195
25%	16.3125	7×7	32.0137	30.5455	45.5098
		9×9	33.3342	30.9273	45.2661
		11×11	34.0708	31.0036	44.1495

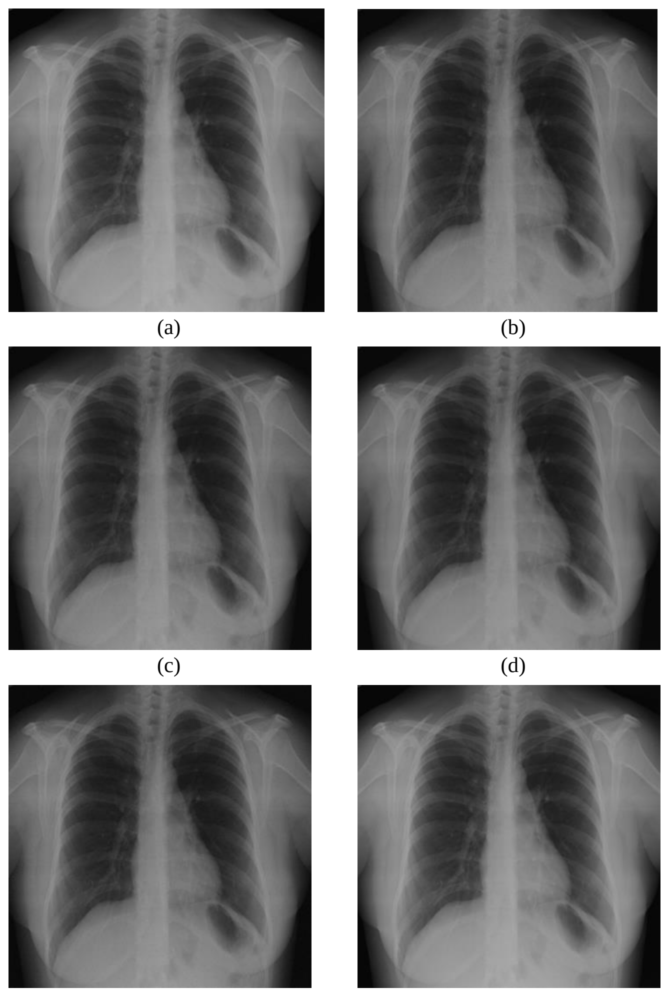
Chest and lung X-Ray image:

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).

TABLE IV
SSIM PERCENT VALUES OF OPTIMIZED FILTERS.

Noise data			Filter		
Level	SSIM	Kernel Size	Mean	Gauss	Median
		3×3	59.3201	59.4820	97.5145
		5×5	73.8881	73.3950	96.0445
10%	26.1078	7×7	80.0669	79.0125	95.3139
		9×9	82.4815	80.0887	94.9174
		11×11	83.4067	81.2753	94.6504
		3×3	48.3178	48.4803	97.2321
		5×5	65.6506	64.9316	95.9766
15%	16.9308	7×7	73.5446	71.6578	95.2698
		9×9	76.7519	72.9867	94.8832
		11×11	78.0389	73.2093	94.6234
		3×3	40.1032	40.2447	96.6667
		5×5	58.7710	57.8933	95.9006
20%	11.7219	7×7	67.8365	65.3275	95.2220
		9×9	71.6773	66.9346	94.8471
		11×11	73.3007	68.0833	94.5919
		3×3	34.1450	34.2658	95.7343
		5×5	53.3073	52.3242	95.8238
25%	8.0893	7×7	63.2273	60.2271	95.1706
		9×9	67.5862	62.8587	94.8050
		11×11	69.5085	64.1283	94.5542



(a) (b) (c) (d) (e) (f)

Fig. 14. Processing of Chest and lung X-Ray 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 did it using the OpenMP.

TABLE V
PSRN VALUES (DB) OF OPTIMIZED FILTERS.

Noise data Level	PSNR	Kernel Size	Filter Mean	Gauss	Median
10%	20.0350	3×3	29.4543	29.3957	50.4992
		5×5	33.5793	33.0090	47.8523
		7×7	35.9817	34.4005	46.2378
		9×9	37.4995	34.8000	45.4176
		11×11	38.4829	34.8804	44.8985
15%	18.3514	3×3	27.7304	27.6728	48.6415
		5×5	31.8051	31.2383	47.7588
		7×7	34.1776	32.6040	46.1992
		9×9	35.6807	32.9949	45.3953
		11×11	36.6673	33.0734	44.8836
20%	17.0729	3×3	26.5007	26.4438	45.7565
		5×5	30.4996	29.9449	47.6548
		7×7	32.7961	31.2705	46.1544
		9×9	34.2348	31.6478	45.3676
		11×11	35.1740	31.7236	44.8635
25%	16.2156	3×3	25.5474	25.4912	42.6366
		5×5	29.4652	28.9248	47.5473
		7×7	31.6746	30.2088	46.1113
		9×9	33.0452	30.5753	45.3394
		11×11	33.9270	30.6481	44.8421

TABLE VI
SSIM PERCENT VALUES OF OPTIMIZED FILTERS.

Noise data Level	SSIM	Kernel Size	Filter Mean	Gauss	Median
10%	20.5374	3×3	59.1579	59.1921	97.5027
		5×5	79.7355	78.4048	96.4102
		7×7	88.1099	85.4020	95.9796
		9×9	91.4174	87.7512	95.7559
		11×11	92.8591	88.9772	95.6168
15%	12.9862	3×3	49.2356	49.2464	97.2201
		5×5	73.7181	72.9807	96.3685
		7×7	84.8214	80.6935	95.9556
		9×9	89.4914	82.4693	95.7400
		11×11	91.5983	84.7696	95.6063
20%	8.7338	3×3	42.0944	42.0947	96.6034
		5×5	68.6288	66.6334	96.3285
		7×7	81.8118	76.5684	95.9369
		9×9	87.6384	80.6704	95.7251
		11×11	90.3543	83.0265	95.5941
25%	6.4832	3×3	36.8985	36.8910	95.4997
		5×5	64.4367	62.2856	96.2804
		7×7	79.1728	74.0838	95.9109
		9×9	85.9831	78.4279	95.7085
		11×11	89.2582	81.0263	95.5819

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 people to give their point of view about the quality of the images. Experts were professionals of different medic 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 VII
MEDICAL EVALUATION AND VALIDATION OF IMAGE PROCESSING

Type	№	Images a	b	c	d	e	f
PELVIS	1	66%	33%				33%
TIBIA	2	66%		33%	33%		33%
CHEST	3	33%		66%	33%		33%

We can see that the percentages of the validation of the 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 3×3, 5×5, 7×7, 9×9, 11×11 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 3×3, 5×5, 7×7, 9×9, 11×11 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 3×3, 5×5, 7×7, 9×9, 11×11 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.

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 5×5. 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 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

[1] S. Sanjay, S. Neeraj, S. Shiru, "Image Processing Tasks using Parallel Computing in Multi core Architecture and its Applications in Medical Imaging". International Journal of Advanced Research in Computer and Communication Engineering Vol. 2, Issue 4, April 2013.

[2] H. Zhu, "Medical image processing Overview," unpublised.

[3] A. S. Y. Bin-Habtoor et al, "Removal Speckle Noise from Medical Image Using Image Processing Techniques," (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 7 (1) , 2016, 375-377.

[4] J. Shiraishi, Q. Li, D. Appelbaum, K. Doi, "Computer-aided diagnosis and artificial intelligence in clinical imaging," In Seminars in nuclear medicine 2011 Nov 1 (Vol. 41, No. 6, pp. 449-462). WB Saunders.

[5] A. Drew. M.D. Torigian, M.A. Fsar, "Radiology Secrets Plis [internet]," Philadelphia: 2017. Chapther 1, Introduction to radiography, fluorodcopy, and tomosynthesis; [cited from 8 the october 2018]; p.3-7. Avaible from: https://www.clinicalkey.es/#!/content/book/3-s2.0-B9780323286381000013

[6] A. Drew. M.D. Torigian, M.A. Fsar, "Radiology Secrets Plis[internet]," Philadelphia: 2017. Chapther 3, Introduction to Nuclear Medicine and Molecular Imaging; [cited from 8 the october 2018]; p.14-19. Avaible from: https://www.clinicalkey.es/#!/content/book/3-s2.0-B9780323286381000037

[7] R.C. Gonzalez, R.E. Woods, "Digital Image Processing," 3rd edition, Prentice-Hall, 2008. ISBN-13: 978-0131687288, 2008.

[8] L. Huang, et al, "Parallelizing Ultrasound Image Processing using OpenMP on Multicore Embedded Systems," 978-1-4673-5085-3/12/$31.00 ©2012 IEEE.

[9] G. Slabaugh, et al., "Multicore Image Processing with OpenMP," unpublished.

[10] S. Patel, "A Survey on Image Processing Techniques with OpenMP," © 2015 IJEDR | Volume 3, Issue 4 | ISSN: 2321-9939.

[11] A. Zotin, K. Simonov, F. Kapsargin, T. Cherepanova, A. Kruglyakov and L. Cadena, "Techniques for Medical Images Processing Using Shearlet Transform and Color Coding," In: Favorskaya M., Jain L. (eds) Computer Vision in Control Systems-4. Intelligent Systems Reference Library, vol 136. Springer, Cham. Chapter First Online: 27 October 2017 DOI https://doi.org/10.1007/978-3-319-67994-5_9.

[12] Chandel et al. Image Filtering Algorithms and Techniques: A Review // International Journal of Advanced Research in Computer Science and Software Engineering 3(10), pp. 198-202, 2013.

[13] B. Gupta, N. Singh, "Image Denoising with Linear and Non-Linear Filters," A Review // International Journal of Computer Science Issues, Vol. 10, Issue 6, No 2, pp. 149-154, 2013.

[14] A. Lukin, "Tips & Tricks: Fast Image Filtering Algorithms," 17-th International Conference on Computer Graphics GraphiCon'2007: 186–189, 2007.

[15] G. Pascal, "A Survey of Gaussian Convolution Algorithms," Image Processing On Line 3: 286–310, 2013.

[16] S. Perreault, P. Hebert, "Median filtering in constant time," IEEE Transactions on Image Processing 16(9): 2389–2394, 2007.

[17] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, R. Menon, "Parallel programing in openmp," Academic Press. USA. 249p ISBN 1-55860-671-8, 2001.

[18] A. Kiessling, "An Introduction to parallel programming with OpenMP," A Pedagogical Seminar. The University of Edinburgh. UK, 2009.

[19] N. M. Thanh and M. S. Chen, "Image Denoising Using Adaptive Neuro-Fuzzy System," IAENG International Journal of Applied Mathematics, vol. 36, no. 1_11, pp 67-73, 2007.

[20] C.U. Lei, C.M. Cheung, and N. Wong, "Efficient 2D Linear-Phase IIR Filter Design and Application in Image Filtering," IAENG International Journal of Applied Mathematics, vol. 37, no. 1_9, pp 56-63, 2007.

[21] A. Horé, D. Ziou, "Image quality metrics: PSNR vs. SSIM," 2010 International Conference on Pattern Recognition. 2010 IEEE. DOI 10.1109/ICPR.2010.579, 1051-4651/10