Parallel Fuzzy Filter for Impulse Noise Removal

J. Arnal\textsuperscript{1}, L. A. Drummond\textsuperscript{2}, L. B. Súcar\textsuperscript{1}, and V. Vidal\textsuperscript{3}

\textsuperscript{1}Departamento de Ciencia de la Computación e Inteligencia Artificial, Universidad de Alicante, San Vicente del Raspeig, Alicante, Spain
\textsuperscript{2}Lawrence Berkeley National Laboratory, Berkeley, CA, USA
\textsuperscript{3}Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Grao de Gandia, Valencia, Spain

Abstract—A parallel algorithm for reducing impulse noise from color images is proposed. The algorithm is based on a fuzzy metric and is performed in two steps followed by noise filtering using the vector median filter. An implementation of the algorithm on multi-core interface using the Open Multi-Processing (OpenMP) is presented. A performance analysis with large images is conducted. Performance is evaluated in terms of execution time and in terms of PSNR. Results show that the proposed filter obtains good performance in terms of PSNR. After applying the multicore optimization strategies, the observed time shows that the proposed filter is able to remove impulse noise in real-time.

Keywords: Parallel computing, OpenMP, Colour image filter, Fuzzy metric, Impulse noise

1. Introduction

Digital images are often corrupted by noise during acquisition and transmission processes. An important problem in image processing is to remove that noise preserving some image features such as edges, textures, and fine details. An specially common type of noise is the impulse noise \cite{1}, \cite{2}. Impulsive noise is commonly caused by the sensor malfunction and other hardware in the process of image formation, storage or transmission \cite{3}. For example, impulsive noise is found in situations where quick transients, such as faulty switching, take place during image processing. This type of noise affects some individual pixels, by changing their original values. The most common noise impulsive model is the Salt and Pepper noise (or fixed value noise), which considers that the new, wrong, pixel value is an extreme value within the signal range. This is the noise type considered in this paper.

Many methods have been introduced to remove impulse noise (see e.g. \cite{3}–\cite{13}).

In \cite{13} a two-step procedure using the fuzzy ROD (ROD was introduced in \cite{14}, and generalized to colour images in \cite{15}) statistic is proposed. In the first step are diagnosed pixels which are clearly noisy or clearly noise-free, and in the second step are diagnosed pixels which are difficult to classify.

Experimental results showed that this filtering technique exhibits competitive results with respect to other state-of-the-art methods. On the other hand, because of the large data set size of high-resolution image data, sequential computers do not have sufficient computing power to perform this algorithm in real-time. Then, this filter has shown good results in quality but does not seem appropriate for real-time processing.

Moreover, this algorithm exhibits a high degree of data locality and parallelism and thus is suitable for parallel computing hardware. Due to these causes, in this paper we introduce a parallel version of fuzzy peer group based on filters introduced in \cite{13} in order to retain their good quality results while trying to improve their performance, so as to make them usable in real-time processing.

We have tested this parallel algorithm developing programs for multi-cores, obtaining a nearly linear speedup as a function of the number of processors used. Nowadays, multi-cores are widely available, and then the introduced approach is an effective, practical, and economical mode for real-time image processing.

This paper is organised as follows: Section 2 explains the proposed parallel noise removal method. Experimental results are shown in Section 3, and finally, the conclusions are presented in Section 4.

2. Parallel noise removal method

Let the color image $A$ be defined as a mapping $\mathbb{Z}^2 \rightarrow \mathbb{Z}^3$. That is, the color image is defined as a two-dimensional matrix $A$ of size $M \times N$ consisting of pixels $x_i = (x_i(1), x_i(2), x_i(3))$, indexed by $i$, which gives the pixel position on the image domain $\Omega$. Components $x_i(l)$, for $i = 1, 2, ..., M \times N$ and $l = 1, 2, 3$, represent the color channel values in RGB quantified into the integer domain.

Let $W$ represents a square filtering window consisting of $n \times n$ color pixels centered at pixel $x_0$. And let $x_i \in W$, $i = 1, \ldots, n^2 - 1$ denote the pixels in the neighborhood of $x_0$. The parallel denoising algorithm introduced in this study uses the fuzzy peer group of a central pixel $x_i$ in a window $W$ according to \cite{16} and using a fuzzy metric. In order to describe the parallel algorithm, and how the pixels were
as,
the
greater for impulse noise pixels than for noise-free pixels.

\[ ROD_m(x) = \sum_{j=1}^{m} r_j(x). \quad (1) \]

Then, \( ROD_m \) expresses the global distance between \( x \) and its \( m \) closest neighbors. This distance is expected to be greater for impulse noise pixels than for noise-free pixels.

![Multi-core](image)

Fig. 1: Image domain decomposition: Distributed image on 4 cores.

assigned to each computing element, we consider a domain decomposition of the image domain \( \Omega \) in \( P \) subdomains \( \{\Omega_i\}_{i=1}^{P} \), where \( P \) is the number of processors. Fig. 1 shows an example of the image domain decomposition used in the experiments.

Fig. 2 shows the parallel filtering algorithm. The method is divided into two stages: noise detection and noise removal. To detect the impulse noise a two steps detection process is used. In the first step, pixels which are clearly noisy or clearly noise-free are classified. The filtering scheme used is based on the FROD statistic described in the following lines.

Consider for each pixel \( x = (x(1), x(2), x(3)) \), in RGB format, a \( n \times n \) window \( W_x \) centered at \( x \). Let \( W_x^0 \) the set of neighbours pixels of \( x \) in \( W_x \), i.e., \( W_x^0 = W_x - \{x\} \). In order to compute the ROD statistic [14] the distances \( d_{x,x_i} \), \( x_i \in W_x^0 \) are ordered in an ascending sequence obtaining a set of non-negative real numbers \( r_j(x) \) such that: \( r_1(x) \leq r_2(x) \leq \ldots \leq r_{n^2-1}(x) \). Then, fixed a positive integer \( m \leq n^2 - 1 \), the \( m \) rank-ordered difference statistic \( ROD_m \) is defined in [14] as,

\[ ROD_m(x) = \sum_{j=1}^{m} r_j(x). \quad (1) \]

To obtain the distances \( d_{x,x_i} \), \( x_i \in W_x^0 \), we use the fuzzy metric \( M_\infty \) [13], that has been proven to be especially sensitive to impulsive noise. This metric, given two RGB color image vectors \( x_i, x_j \), is defined by

\[ M_\infty(x_i, x_j) = \sum_{l=1}^{3} \min \{x_{il}, x_{jl}\} + K \]

We have set \( K = 1024 \) which has been proved to be an appropriate value for RGB colour vectors [17].

Considering the usage of the \( M_\infty \) fuzzy metric to obtain the distances \( d_{x,x_i} \), \( x_i \in W_x^0 \), the fuzzy ROD (FROD) statistic is defined as follows. Taking the fuzzy distances \( d_{x,x_i} \) ordered in a descending sequence \( s_1(x) \geq s_2(x) \geq \ldots \geq s_{n^2-1}(x) \) the \( FROD_m \) statistic is defined by

```plaintext
Require: Image \( A \), a domain decomposition \( \{A_{\Omega_k}\}_{k=1}^{P} \),
\( th_3, th_2, th_3 \)
Ensure: Filtered image.
1: for \( k = 1, \ldots, P \), in parallel do
2: Impulse noise detection: Step 1
3: for \( x_i \) pixel in \( A_{\Omega_k} \) do
4: Processor \( k \) calculates: \( d = FROD_m(x_i) \);
5: if \((d > th_1)\) then
6: pixel \( x_i \) and \( \forall x_j \) used in \( FROD_m(x_i) \) are
7: classified as noise-free;
8: else
9: if \((d < th_2)\) then
10: \( x_i \) is classified as noisy;
11: else
12: \( x_i \) is classified as non-diagnosed;
13: end if
14: end if
15: Impulse noise detection: Step 2
16: for \( x_i \) pixel in \( A_{\Omega_k} \) classified as non-diagnosed in
17: Step 1 do
18: Processor \( k \) calculates \( d = FROD_m(x_i) \) exclud-
19: ing the pixels previously classified as noisy;
20: if \((d > th_3)\) then
21: pixel \( x_i \) and \( \forall x_j \) used in \( FROD_m(x_i) \) are
22: classified as noise-free;
23: else
24: \( x_i \) is classified as noisy;
25: end if
26: end for
27: Impulse noise reduction:
28: for \( x_i \) pixel in \( A_{\Omega_k} \) classified as noisy do
29: \( x_i \) is replaced with VMF_out
30: end for
```

Fig. 2: Parallel filtering algorithm.
Another filter parameter $m$ is used. In this step a third threshold parameter $\text{FROD}_m(x)$, $x$ is classified as noisy. If $x$ satisfies $\text{FROD}_m(x) > \theta_3$, then we conclude that it is not possible to classify $x$ at this step, and it is analyzed in a second step. In the second step, a third threshold parameter $\theta_3$ is used. In this step $\text{FROD}_m(x)$ is computed on $W^0_x$ excluding the pixels previously classified as noisy, and using another filter parameter $m' < m$. If $\text{FROD}_{m'}(x) > \theta_3$, then $x$ and its $m'$ neighbours involved in the computation of $\text{FROD}_{m'}(x)$ are classified as noise-free. Otherwise, $x$ is classified as noisy.

After the noise detection steps in the noise reduction stage, each pixel classified as noisy is replaced with VMF$_{out}$ [18] operating over its noise-free neighbours in a $n \times n$ window.

### 3. Experimental Results

We carried out specific experiments and developments using two different machines and software settings which are included in the following list:

- Multi-core 1: Intel Xeon CPU E5320 (8 cores), 1.86 GHz, 8GB RAM, Linux Ubuntu 8.04.1. GNU Fortran compiler.
- Multi-core 2: Intel XEON X5660 (12 cores), 2.8 GHz, 48 GB RAM, Linux CENTOS 5.6. Intel Fortran Compiler.

Different test images shown in Fig. 3 were used in the experiments: Lenna [4], Caps [19], Motorbikes [19], Statue [19], Bus [20], and Toy [20]. These images have been corrupted with impulse noise. The random-value impulse noise [1] was considered. We denote by $p$ the noise appearance probability. In our tests we have used $p \in [0, 0.1]$.

Fig. 1 shows an example of the image domain decomposition used in the experiments using 4 cores. In order to adjust the filter parameters $\theta_1$, $\theta_2$, and $\theta_3$ in [13] the filter performance was analyzed in terms of Peak Signal to Noise Ratio (PSNR), as a function of $\theta_1$, $\theta_2$, and $\theta_3$ contaminating images with different probabilities of impulse noise $p$. Accordingly to that study, our results were obtained setting $\theta_1$, $\theta_2$, and $\theta_3$ proportionally to $p$ as follows.

\[
\begin{align*}
\theta_1 &= 0.90 + \frac{p}{0.4} 0.07, \\
\theta_2 &= 0.87 + \frac{p}{0.4} 0.06, \\
\theta_3 &= 0.97 + \frac{p}{0.4} 0.01,
\end{align*}
\]

According to previous research [13], in the experiments we have considered a $3 \times 3$ filter window $(n = 3)$ and $m = 3$, $m' = 1$.

We designed both the serial code and parallel code and then compared the execution time.

Tables 1–4 show the results obtained on Multi-core 2 dividing the image among different number of cores and Fig. 4, presents the speedup obtained for different sizes of Caps and Statue images. To quantify parallel performance, parallel speedup $S_p$ is computed as:

\[
S_p = \frac{T_{seq}}{T_P}
\]

where $T_{seq}$ is the execution time of the sequential algorithm and $T_P$ is the execution time of the parallel algorithm using $P$ processors. The results show that a significant speedup is achieved. On the other hand, Figs. 4 shows that the optimal number of processors to filter the image depend on the image size.

The filter performance has been evaluated using the Peak Signal to Noise Ratio (PSNR), that measures the noise suppression capability [1]. Fig. 5 shows that PSNR performance improves as image size increases. From the visual point of view, by inspecting the denoised images in Fig. 6, it can be concluded that the filter obtains robust results. $\text{FROD}_m$ consistently diagnose and reduce impulses while preserving the quality of image edges and details.

### 4. Conclusion

A parallel algorithm based on a fuzzy metric has been proposed to detect and remove impulse noise in digital images. We have implemented it on multi-cores using OpenMP. The parallel algorithm introduced demonstrated a significant speedup in processing large images compared to sequential algorithm. The algorithm obtained a nearly linear speedup as a function of the number of processors used. Multi-cores are widely available, so the introduced approach is an effective, practical, and economical mode of real-time image processing.

### References


Fig. 3: Test images: (a) Lenna, (b) Caps [19], (c) Motorbikes [19], (d) Statue [19], (e) Bus [20], and (f) Toy [20].

Table 1: Computational time in seconds on Multi-core 2. Caps image. 1600 × 1067 pixels

<table>
<thead>
<tr>
<th>Impulse Noise</th>
<th>Number of processors</th>
<th>PSNR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 4 6 8 10 12</td>
<td></td>
</tr>
<tr>
<td>$p = 0.05$</td>
<td>0.481173992 0.243747950 0.124355078 0.085099936 0.068102121 0.054935932 0.054033995</td>
<td>39.05</td>
</tr>
<tr>
<td>$p = 0.1$</td>
<td>0.69497004 0.351624966 0.178093195 0.11976024 0.091516972 0.075078964 0.064303875</td>
<td>32.90</td>
</tr>
</tbody>
</table>

Table 2: Computational time in seconds on Multi-core 2. Motorbikes image. 1600 × 1067 pixels

<table>
<thead>
<tr>
<th>Impulse Noise</th>
<th>Number of processors</th>
<th>PSNR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 4 6 8 10 12</td>
<td></td>
</tr>
<tr>
<td>$p = 0.05$</td>
<td>0.475349188 0.240406036 0.122515917 0.084091902 0.06705308 0.055345058 0.064940929</td>
<td>36.30</td>
</tr>
<tr>
<td>$p = 0.1$</td>
<td>0.685279131 0.34661603 0.175352097 0.118137121 0.091350079 0.075542927 0.06315589</td>
<td>30.67</td>
</tr>
</tbody>
</table>

Table 3: Computational time in seconds on Multi-core 2. Bus image. 3986 × 2538 pixels

<table>
<thead>
<tr>
<th>Impulse Noise</th>
<th>Number of processors</th>
<th>PSNR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 4 6 8 10 12</td>
<td></td>
</tr>
<tr>
<td>$p = 0.05$</td>
<td>2.842068911 1.434518814 0.723886013 0.485067129 0.367190838 0.300970078 0.271684885</td>
<td>37.51</td>
</tr>
<tr>
<td>$p = 0.1$</td>
<td>4.100522995 2.073998928 1.045986891 0.700412035 0.528025866 0.433892012 0.377089977</td>
<td>31.68</td>
</tr>
</tbody>
</table>


Table 4: Computational time in seconds on Multi-core 2. Toy image. 3551 × 2160 pixels

<table>
<thead>
<tr>
<th>Impulse Noise</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>PSNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>p = 0.05</td>
<td>1.905487064</td>
<td>0.956923962</td>
<td>0.513839006</td>
<td>0.354380879</td>
<td>0.270764112</td>
<td>0.221177816</td>
<td>0.227982998</td>
<td>39.74</td>
</tr>
<tr>
<td>p = 0.1</td>
<td>2.467350996</td>
<td>1.232701063</td>
<td>0.655728817</td>
<td>0.454741001</td>
<td>0.346143007</td>
<td>0.28381896</td>
<td>0.243230104</td>
<td>32.02</td>
</tr>
</tbody>
</table>

Fig. 4: Speedup for different image sizes. Caps Image corrupted with impulse noise p = 0.1.

(a) Motorbikes image (b) Caps image (c) Statue image

Fig. 5: PSNR for different image sizes. Images corrupted with impulse noise p = 0.1.

Fig. 6: Filter outputs for visual comparison: (a) Statue image, (b) image corrupted with $p = 0.05$ impulse noise, (c) filter output, (d) Lenna image, (e) image corrupted with $p = 0.05$, and (f) filter output.