A Parallel Edge Preserving Algorithm for Salt and Pepper Image Denoising

M. Aldinucci[1], C. Spampinato[2] M. Drocco[1], M. Torquati[3], S. Palazzo[2]
[1] Computer Science Dept. - University of Torino, Italy
e-mail: aldinuc@di.unito.it, maurizio.drocco@gmail.com
[2] Dept. of Electrical, Electronics and Computer Engineering - University of Catania, Italy
e-mail: {cspampin, palazzosim}@dieei.unict.it
[3] Computer Science Dept. - University of Pisa
e-mail: torquati@di.unipi.it

***Abstract*— In this paper a two-phase filter for removing "salt and pepper" noise is proposed. In the first phase, an adaptive median filter is used to identify the set of the noisy pixels; in the second phase, these pixels are restored according to a regularization method, which contains a data-fidelity term reflecting the impulse noise characteristics. The algorithm, which exhibits good performance both in denoising and in restoration, can be easily and effectively parallelized to exploit the full power of multi-core CPUs and GPGPUs; the proposed implementation based on the FastFlow library achieves both close-to-ideal speedup and very good wall-clock execution figures.**

***Keywords*— Impulse Denoising, Image Restoration, Multi-core parallelization, lock-free synchronization**

I. Introduction

The ever increasing computing power available from off-the-shelf processors has allowed researchers to extend the number of applications in image processing and machine vision. One important step in any machine vision system is the image restoration phase, which has gathered the attention of image processing researchers, especially with the massive production of digital images and movies, often grabbed in poor conditions. A typical noise that affects digital images is "Salt and Pepper" noise [1], which may be caused by malfunctioning pixels in camera sensors, faulty memory locations in hardware or transmission in a noisy channel [2]. This noise sets the corrupted pixel value to the maximum or the minimum of the pixels variation range (0 or 255 for an 8-bit image). During the last fifteen years, a large number of methods have been proposed to deal with salt and pepper noise (and more in general impulse noise) from digital images [3]. Most of these methods are order statistic filters that exploit the rank-order information of an appropriate set of noisy input pixels. The median filter is the most popular non-linear filter for removing impulse noise, because of its good denoising power [2] and its computational efficiency [4], but it affects image details while removing noise. These issues have been generally addressed by filtering techniques based on the median filter modifications [5], [6]. However, the performance of median filtering based approaches is unsatisfactory in suppressing signal-dependent noise [7] when the noise percentage is high (more than 50%). To achieve a good compromise between the image-detail preservation and the noise reduction an impulse detector must be used before filtering. Several types of impulse detectors exist: the most famous is the progressive switching median (PSM) [8]. Machine learning approaches have also been widely used in the last years, e.g. approaches relying on neural networks [9], Bayesian networks [10], fuzzy logic [11] and neuro-fuzzy [12]. The filtering is then selectively applied to the noisy regions detected by the noise detector. To the best of our knowledge, one the most effective algorithm for edge preserving in salt and pepper denoising has been proposed by Nikolova in [13] that applies a variational method for image details preserving that is based on a data-fidelity term related to the impulse noise. Based on this approach Chan *et al.* in [14] (called for simplicity Chan's method) proposed a powerful filter able to remove salt and pepper noise as high as 90%. Similar approaches to the Chan's method, aiming at improving the noisy detection step and at reducing the processing times, are the ones proposed in [10], [15]–[18]. In this paper an optimized version of the Chan's method is introduced and tested with various test images. Moreover, the algorithm has been parallelized both using FastFlow [19], a framework for parallel programming over multicore platforms, and GPU programming for improving the efficiency of the filter in order to be really compatible with real-time applications.
The outline of the paper is as follows: in the next section the denoising filter is reviewed. Section 3 shows the performance of the sequential implementation of the described filter, whereas sect. 4 and section 5 point out, respectively, the parallel implementation of the proposed algorithm and the experimental results. Finally, in the last section concluding remarks are given.

II. Two-phase Edge Preserving Filter

The proposed filter for impulse noise is a two-phase algorithm: 1) Noisy pixels Identification (called *detect*) and 2) Noisy Pixel Restoration (called *denoise*). The first step identifies noisy pixels by means of a modified Adaptive Median Filter (AMF) classifier; whereas the second step restores them using a variational approach [13].

978-1-4673-2584-4/12/$31.00 ©2012 IEEE

A. Noisy Pixels Identification via Adaptive Median Filter

Let $\hat{y}$ be the map of noisy pixels, obtained by applying the adaptive median filter classifier to the noisy image, that has 1 in correspondence of the position of the noisy pixels, whereas 0 in correspondence of the uncorrupted pixels. Hence the set of noisy pixels N (where the restoration algorithm has to be applied) consists of the overall pixels of the original image y whose values in the $\hat{y}$ map are equal to 1, i.e.: $N = \{(i,j) \in \mathrm{A} : \hat{y}_{i,j} = 1\}$

The set of all uncorrupted pixels is $N^c = A \setminus N$, where A is the set of all pixels. Notice that, since the pixels with color different from 0 or from 255 are surely uncorrupted, the AMF filter [20] can be modified to exclude them with significant benefits in term of performance and false positive rate.

B. Variational Method for noisy pixels restoration

The problem of image restoration for edge preserving is an inverse problem solved by using regularization, where the restored image u is given by solving the following optimization problem restricted to the set of the noisy pixels N.

$$\min_{u \in N} F(u) = \alpha \int R(u) + \beta \int D(u,d) \tag{1}$$

Where d is the image corrupted by the noise, $D(u,d)$ is the data-fidelity term that is related to the kind of noise and provides a measure of the deviation between d and the output image u, whereas $R(u)$ is a regularization term that uses a-priori knowledge for enforcing the solution and should be represented by a function that penalizes/removes only irregularities due to the affecting noise, thus leaving out high-level discontinuities (edges). β and α are the regularization parameters that balance the effects of both mentioned terms. Among all the functionals F(u) (see [21]) for edge preserving proposed during the last fifteen years, we have selected the one proposed in [14]:

$$F_d|_N(u) = \sum_{(i,j) \in N} [|u_{i,j} - d_{i,j}| + \frac{\beta}{2}(S_1 + S_2)] \tag{2}$$

where

$$S_1 = \sum_{(m,n) \in V_{i,j} \cap N} 2 \cdot \varphi(u_{i,j} - d_{m,n}) \tag{3}$$

$$S_2 = \sum_{(m,n) \in V_{i,j} \cap N^c} \varphi(u_{i,j} - u_{m,n}) \tag{4}$$

where N represents the noisy pixels set, N^c the set of uncorrupted pixels, and $V_{i,j}$ is the set of the four closest neighbors of the pixel with coordinates (i,j) and d is the corrupted image. As in [14], we have used the following φ function that provides the best trade-off between edge preserving and denoising: $\varphi(t) = |t|^\alpha$ $\varphi(t) = |t|^\alpha$ with $1 < \alpha \leq 2$. The values of α and β were, respectively, set to 1.3 and 4 in order to guarantee the trade-off between noise removal and edge preservation provided by the function φ. The minimization problem is then solved by an algorithm that works on the residuals $z = u - y$ of the functional (1) and it is following reviewed:

– Initialize $z_{ij}^{(0)} = 0$ for each $(ij) \in A$;
– At each iteration k, calculate, for each $(ij) \in A$,

$$\xi_{i,j}^{(k)} = \beta \sum_{(m,n) \in V_{i,j}} \dot{\varphi}\left(y_{i,j} - z_{i,j} - y_{m,n}\right)$$

where $z_{m,n}$, for $(m,n) \in Vi,j$, are the latest updates and $\dot{\varphi}$ is the derivative of φ.
– If $\xi_{i,j}^{(k)} \leq 1$, $z_{i,j}^{(k)}$ will be set to 0. If not, $z_{i,j}^{(k)}$ is the solution of the following equation:

$$\beta \sum_{(m,n) \in V_{i,j}} \dot{\varphi}\left(z_{i,j}^{(k)} + y_{i,j} - z_{m,n} - y_{m,n}\right) = sign\left(\xi_{i,j}^{(k)}\right)$$

The quasi-Newton method [22] is recursively applied to find the restored image $\hat{u}$ that minimizes the functional shown in (2). Examples of convergence criteria are:

$$\left|\left|z^{(k)}\right| - \left|z^{(k-1)}\right|\right| < \epsilon, \epsilon \in \mathbb{N} \tag{5}$$

$$\frac{\sum_{(i,j) \in N} \left|\left|z_{i,j}^{(k)}\right| - \left|z_{i,j}^{(k-1)}\right|\right|}{|N|} < \epsilon, \epsilon \in \mathbb{R} \tag{6}$$

III. Sequential Algorithm: Experimental Evaluation

The algorithm has been prototyped in C++ and tested on a single core of an Intel workstation Xeon E7-4820 @2.0GHz with 8MB L3 cache and 64 GBytes of main memory with Linux x86_64. The test of the denoising filter was performed on standard testing images of different sizes (see Fig. 1). In order to compare results with literature, the results reported in this section are referred to 256x256 8-bit grayscale images: Lena and Peppers featured by homogeneous regions and Bridge and Baboon characterized by high activity (see Fig. 1). These images have been corrupted by "salt" (value 255) and "pepper" (value 0) noise with equal probability; a range of noise levels, from 10% to 90%, was tested.

The algorithm was evaluated according to two main metrics: 1) noise identification and restoration quality, and 2) execution time. Denosing and restoration performance was measured by the peak signal-to-noise ratio (PSNR) and the mean absolute error (MAE) defined as follows:

$$PSNR = 10 \cdot \log_{10} \frac{255^2}{\frac{1}{MN} \sum_{i,j} (r_{i,j} - x_{i,j})^2}$$

$$MAE = \frac{1}{MN} \sum_{i,j} (r_{i,j} - x_{i,j})^2$$

where $r_{i,j}$ and $x_{i,j}$ denote, respectively, the pixel values of the restored image and the original image. Table 1 shows the PSNR and MAE, respectively, for Lena, Peppers, Bridge and

Fig. 1. Images used for testing the proposed denoising algorithm (Lena, Peppers, Bridge, Baboon, and Space).

	Lena — Noise level					Peppers — Noise level					Bridge — Noise level					Baboon — Noise level				
	10%	30%	50%	70%	90%	10%	30%	50%	70%	90%	10%	30%	50%	70%	90%	10%	30%	50%	70%	90%
PSNR	41.96	35.76	32.56	29.47	24.43	42.27	35.49	31.57	28.23	23.12	36.61	31.50	28.11	25.23	21.22	35.39	29.73	26.94	24.61	22.06
MAE	0.36	1.21	2.27	3.85	8.82	0.30	1.10	2.18	3.94	9.90	0.82	2.61	4.85	8.06	15.01	0.94	3.14	5.63	8.87	13.96

Table 1. PSNR and MAE obtained at varying of the noise affecting Lena, Peppers, Bridge and Baboon images.

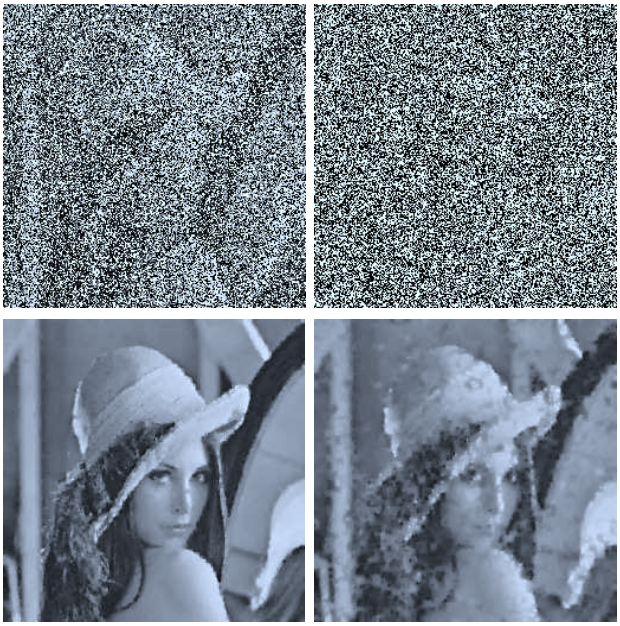

Fig. 2. Lena: 70% & 90% of noise and restored version.

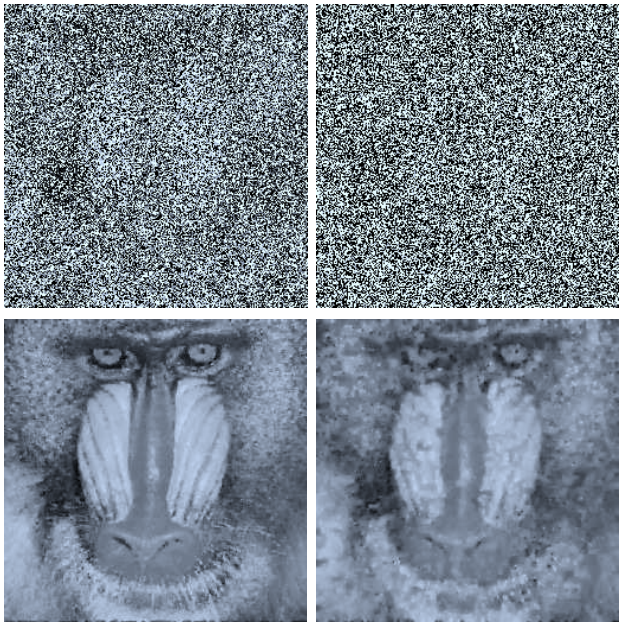

Fig. 3. Baboon: 70% & 90% of noise and their restored versions.

Baboon images reported at varying of the noise level (from 10% to 90% at step of 20%).

To better estimate the achieved restoration quality, Fig. 2 and Fig. 3 show the denoised images (Lena and Baboon) while they are affected, respectively, by 70% and 90% of noise. Fig. 4 shows the denoised Space image, which is quite more complex due to bright spots and dark background.

The first phase (detect) exhibits a linear execution time with respect to the total number of pixels. However, for the tested images, the detect phase is two–three orders of magnitude faster than a single iteration of the second step (denoise). The second phase shows a computational complexity of $\mathcal{O}(n\_noisy\_pixels \cdot n\_iterations)$, where n_noisy_pixels is the number of noisy pixels identified in the first step, and $n\_iterations$ is the number of iteration required to reach one of the convergence criterions (see section 2).

Although the denoising performance, in terms of PSNR and MAE, of the proposed approach is comparable or better with the existing variational based approaches, e.g. Chan's method [14], Cai's method [10], [16] , Chen's method [17], the proposed algorithm appears significantly more efficient with

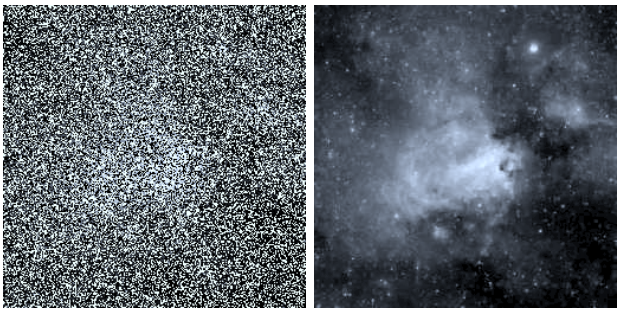

Fig. 4. Space: 70% of noise and its restored version.

respect to the above methods (e.g. 75s for Lena 256x256 with 90% of noise, PNSR=24.42, MAE=8.81).

We believe that the major factor of speed of the proposed prototype is due mainly to a number of optimizations we performed on of basic algorithm, such as reduction of expensive mathematic operations (thanks to cycle invariants), better memory management and noisy pixel manipulation.

IV. Parallel Implementation

The achieved performance (see Sec. III) might be still considered not sufficient for real-time applications (e.g. video) and large images. Moreover, the sequential prototype of the algorithm do not exploit neither the full power of multi-core platforms not the potential of GPGPUs accelerators. We advocate a FastFlow-based high-level parallelization of the algorithm that is able to exploit parallelism in both phases and on both multi-core and GPGPUs. The high-level approach make it possible to realize the parallel porting with a low programming effort.

FastFlow is a C++ pattern-based parallel programming framework aimed at simplifying the development of applications for (shared-memory) multi-core and GPGPUs platforms. The key vision of FastFlow is that ease-of-development and runtime efficiency can both be achieved by raising the abstraction level of the design phase, thus providing developers with a set of parallel programming patterns, such as *farm*, *divide&conquer*, *pipeline*, *map*, *reduce* patterns, and supports their arbitrary nesting and composition. [19]. Map and reduce patterns can be run both on multi-cores and offloaded onto GPGPUs. In the latter case, the business code can include GPGPU-specific statements (i.e. CUDA or OpenCL statements). FastFlow is available as an open source software under LGPLv3 at FastFlow website [23], where are also reported performance comparison against other programming tools such as POSIX, Cilk, OpenMP, and Intel TBB.

A. The high-level algorithm

The two-phase fiter methodology naturally induces an high-level structure for algorithm, which is can be described as the successive applications of two filters as described in Fig. 5. The two phases can operate in pipeline in the case the two-phase denoiser is used on a stream of images (e.g. in a video application). In addition, both filters can be parallelized in a data-parallel fashion. Let **map** $f[a_0, a_1, \ldots] = [f(a_0), f(a_1), \ldots]$ and **reduce** $\oplus\ [a_0, a_1, \ldots] = a_0 \oplus a_1 \oplus \ldots$, where $\oplus$ is a binary associative operator, and $[a_0, a_1, \ldots]$ an array (e.g. the pixels of an image):

a) Detect: : in the detect step (i.e. an adaptive median filter) each pixel can be processed independently, provided the processing element can access to a read-only halo of the pixel is available. The process is expressed by way of the **map** pattern, which applies the following blocks of code to all pixels. A set of (candidate) noisy pixels is produced.

b) Denoise: : The denoise step follows a similar approach, as the variational method can be computed independently for each pixel (with previous iteration halo). This step is iterated up to a convergence criteria is reached. The convergence criteria is a global propriety of the image, thus should be computed via a **reduce** parallel operator by reducing the difference between last two iterations according to a given convergence criteria (e.g. based on PSNR as 5 or MAE as 6).

B. Multi-core and GPGPUs variants

The high-level approach significantly ease code portability and performance portability onto different platforms. The approach is fully supported by the FastFlow framework that provides the programmer with both multi-core and GPGPUs (efficient) implementation of the parallel patterns (i.e. map, reduce, farm, pipeline). In particular, the multi-core implementations of patterns are realized via non-blocking graphs of threads connected by way of lock-free channels [19], whereas the GPGPUs implementation are realized by way of C++/CUDA templates and offloading technique [24]. Also, different pattern can be mapped onto different sets of cores or accelerators, thus suing the full available power of multi-core platforms equipped with accelerators. The business code running onto GPGPUs can be further hand-tuned by exploiting CUDA-specific features, e.g. data buffering in the device fast memory (e.g. so-called *shared memory* in the CUDA framework). This kind of fine-tuning might bring significant performance improvements and, in general, can be hardly automatized at the FastFlow level because is typically very related to the business code. However, the performance gain that can be achieved from this kind of tuning is expected to decrease with novel GPGPUs boards supporting hardware caching (e.g. latest NVidia Tesla GPUGPUs with CUDA 4.x).

V. Experimental Evaluation

All experiments reported in this section have been conducted on an Intel workstation with 4 eight-core double-context Xeon E7-4820 @2.0GHz, 18MB L3 shared cache, 256K L2, and 64 GBytes of main memory equipped with a NVidia Tesla C2050 GPGPU with Linux x86_64, gcc 4.4, FastFlow 1.1 and CUDA 4.0.

Fig. 6 shows the speedup achieved with multi-core variant of the algorithm on Lena image 512x512 and Space image

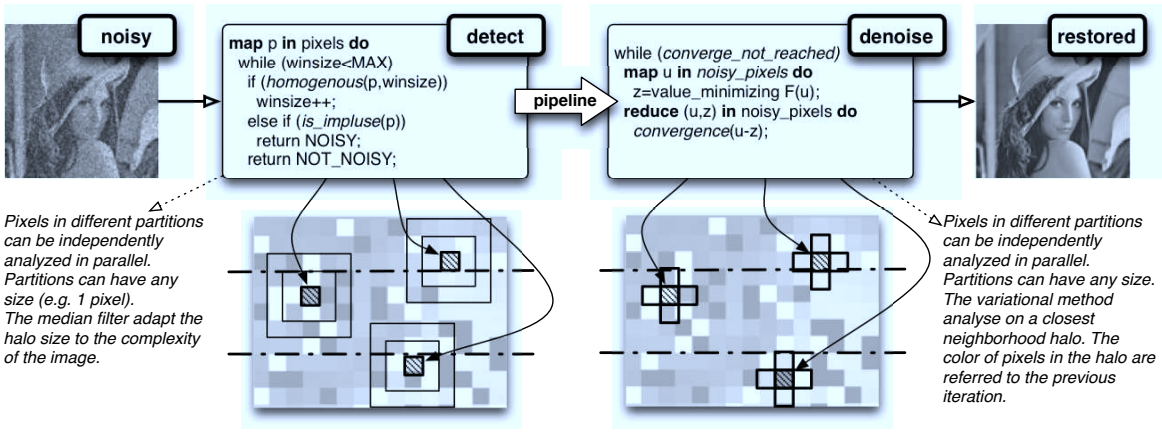

Fig. 5. Two-phase denoiser.

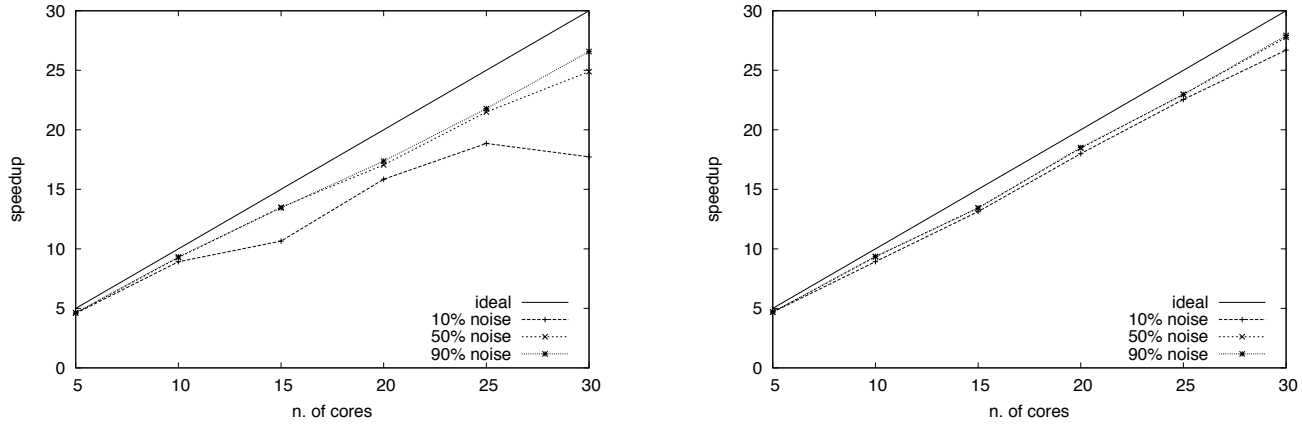

Fig. 6. Speedup on Xeon E7-4820 32 cores platform: left) Lena 512x512; right) Space 4096x4096.

4096x4096 for different levels of noise. In general, as the 90% of the total time is spent in denoising phase, the multi-core variant exhibits an almost ideal speedup in all configurations for non-trivial number of noisy pixels (i.e. large images or quite noisy small images). However, it maintains a decent speedup also for very fine computational grain. As an example, as shown in Fig. 6 left, the denoising of a 512x512 image with 10% of noise using 30 cores still keeps a $17X$ speedup with a execution time per core of only 56 ms.

Execution wall-clock time are shown in Table 2 for both multi-core and multi-core with GPGPU variants. The Lena image of size 512x512 with 90% of noise can be restored in about 10.9 seconds (using all cores) versus the 290 seconds required by the sequential version, resulting in more than $26X$ speedup. Increasing the image size the speedup increase up to $28X$ for almost all noise levels. The speedup grows to $104X$ in the case the most expensive phase (denoising) is offloaded to the GPGPU Tesla accelerator. Thanks to FastFlow very low synchronization overhead, similar results can be achieved also in larger platform configurations. Also, the experimentation

Table 2. Performance for the Lena image (512x512) in two configurations: FastFlow on CPU only (32cores), and FastFlow on CPU and GPGPUs.

noise	FF 32cores (s)	FF 8cores + Tesla (s)	Seq (s)
10	1.8	1.9	32
50	6.5	2.3	162
90	10.9	2.8	290

on different images shown that the performance are quite independent from image complexity since this principally affect the computational time of detect phase, which is not the bottleneck of the algorithm.

WIth respect to the quality of restoration, the low rate of false positives in the detection of noisy pixels achieved by the adaptive median filter brings an overall improvement of the quality of restoration as non-noisy pixels are not changed by the denoise phase. We noticed a improvement of the MAE (see Table 3) with respect to the original version of the algorithm (Chan's method in [14]) especially for high levels of noise.

Table 3. Comparison between our approach and the Chan's one [14] in terms of PSNR and MAE for Lena and Bridge images.

	Method	Lena — Noise level				Bridge — Noise level			
		30%	50%	70%	90%	30%	50%	70%	90%
PSNR	Chan's	35.20	32.21	29.03	22.46	30.29	27.91	25.00	19.02
PSNR	Our	35.76	32.56	29.47	24.43	31.50	28.11	25.23	21.22
MAE	Chan's	1.5	2.97	4.2	12.45	3.75	5.30	8.10	21.25
MAE	Our	1.21	2.27	3.85	8.82	2.61	4.85	8.06	15.01

VI. Concluding Remarks

We have proposed a novel approach for parallel edge preserving restoration of images (and videos) affected by salt and pepper noise. The approach builds on existing techniques by conjugating their strengths. As result, the proposed method is able to achieve a restoration quality —in term of visual quality, PSNR and MAE— better or similar to the best existing approaches and that is particularly fast and scalable on parallel architectures (both multi-core and GPGPUs).

The restoration is made in two phases: detect and denoise. The detect phase, which is based on a adaptive median filter is both effective and fast in identifying candidate noisy points. On salt and pepper noise It is able to provide a close to zero rate of false positives and can be easily parallelized. It can be also used to restore the image, but the filtering does not preserve the edges of the image and the restoration quality is often disappointing. For this, the noisy points are restored in a successive phase by way of a variational method (denoise), which provides a high-quality edge-preserving restoration even if has an higher computational cost. The application of the variational method to noisy point only (unlike other approaches base on the same technique, e.g. [14]), make it possible to both improve the restoration quality and speed. To further speed up the denoising phase, which remains expensive, it has been parallelized for both multi-core and GPGPUs. To the best of our knowledge no other algorithm for removing salt and pepper noise can currently compete in terms of overall quality of restoration and execution time with the two-phase filter proposed in this paper.

The parallel version has been semi-automatically derived by the sequential algorithm thanks to FastFlow and its software acceleration technique [24]. The possibility of pipelining the two phases make it possible to further accelerate the denoising of stream of images (e.g. video applications). Despite the very limited development effort required, the parallel version guarantees close to optimal speedup and scalability on standard cache-coherent multi-core workstations. Also, we believe that the denoising problem is a paradigmatic example of image processing applications, and therefore the proposed parallelization approach can be easily and successfully extended to many other algorithms.

References

[1] R. Gonzales and R. Woods, *Digital image processing*. Prentice Hall, 2002.

[2] A. Bovik, *Handbook of Image and Video Processing*. Academic Press, 2000.

[3] J. Astola and P. Kuosmanen, *Fundamentals of Nonlinear Digital Filtering*. Boca Raton, CRC, 1997.

[4] T. S. Huang, G. J. Yang, and G. Y. Tang, "A fast two-dimensional median filtering algorithm," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 27, no. 1, 1979.

[5] L. Yin, R. Yang, M. Gabbouj, and Y. Neuvo, "Weighted median filters: a tutorial," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 43, Mar 1996.

[6] T.-C. Lin, "A new adaptive center weighted median filter for suppressing impulsive noise in images," *Inf. Sci.*, vol. 177, no. 4, 2007.

[7] P. Yang and O. Basir, "Adaptive weighted median filter using local entropy for ultrasonic image denoising," in *Image and Signal Processing and Analysis, 2003. ISPA 2003. Proc. of the 3rd Intl. Symposium on*, vol. 2, Sept. 2003.

[8] Z. Wang and D. Zhang, "Progressive switching median filter for the removal of impulse noise from highly corrupted images," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 46, Jan 1999.

[9] A. Faro, D. Giordano, and C. Spampinato, "Neural network combined with fuzzy logic to remove salt and pepper noise in digital images," in *Applications of Soft Computing*, vol. 36 of *Advances in Soft Computing*, Springer Berlin / Heidelberg, 2006.

[10] A. Faro, D. Giordano, G. Scarciofalo, and C. Spampinato, "Bayesian networks for edge preserving salt and pepper image denoising," in *Image Processing Theory, Tools and Applications, 2008. IPTA 2008. First Workshops on*, Nov. 2008.

[11] H. Xu, G. Zhu, H. Peng, and D. Wang, "Adaptive fuzzy switching filter for images corrupted by impulse noise," *Pattern Recognition Letters*, vol. 25, no. 15, 2004.

[12] M. Yuksel, "A hybrid neuro-fuzzy filter for edge preserving restoration of images corrupted by impulse noise," *IEEE Transactions on Image Processing*, vol. 15, Apr. 2006.

[13] M. Nikolova, "A variational approach to remove outliers and impulse noise," *Journal of Mathematical Imaging and Vision*, vol. 20, no. 1, 2004.

[14] R. Chan, C. Ho, and M. Nikolova, "Salt-and-pepper noise removal by median-type noise detectors and detail-preserving regularization," *IEEE Trans. on Image Processing*, vol. 14, October 2005.

[15] J.-F. Cai, R. H. Chan, and C. Fiore, "Minimization of a detail-preserving regularization functional for impulse noise removal," *J. Math. Imaging Vis.*, vol. 29, no. 1, 2007.

[16] J.-F. Cai, R. Chan, and M. Nikolova, "Fast two-phase image deblurring under impulse noise," *Journal of Mathematical Imaging and Vision*, vol. 36, 2010.

[17] S. Chen and X. Yang, "A variational method with a noise detector for impulse noise removal," in *Scale Space and Variational Methods in Computer Vision*, vol. 4485 of *LNCS*, Springer, 2007.

[18] F. Cannavò, D. Giordano, G. Nunnari, and C. Spampinato, "Variational method for image denoising by distributed genetic algorithms on grid environment," in *Proc. of the 15th IEEE Intl. Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2006)*, 2006.

[19] M. Aldinucci, M. Danelutto, P. Kilpatrick, and M. Torquati, "Fastflow: high-level and efficient streaming on multi-core," in *Programming Multi-core and Many-core Computing Systems* (S. Pllana and F. Xhafa, eds.), Parallel and Distributed Computing, ch. 13, Wiley, 2012.

[20] H. Hwang and R. A. Haddad, "Adaptive median filters: new algorithms and results," *IEEE Trans Image Process*, vol. 4, 1995.

[21] R. Chartrand and V. Staneva, "Total variation regularisation of images corrupted by non-gaussian noise using a quasi-newton method," *Image Processing, IET*, vol. 2, pp. 295 –303, Dec. 2008.

[22] J. F. Bonnans, J. C. Gilbert, C. Lemaréchal, and C. A. Sagastizábal, *Numerical Optimization: Theoretical and Practical Aspects (Universitext)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[23] M. Aldinucci and M. Torquati, *FastFlow website*, 2009. `http://mc-fastflow.sourceforge.net/`.

[24] M. Aldinucci, M. Danelutto, P. Kilpatrick, M. Meneghin, and M. Torquati, "Accelerating code on multi-cores with FastFlow," in *Proc. of 17th Intl. Euro-Par 2011 Parallel Processing* (E. Jeannot, R. Namyst, and J. Roman, eds.), vol. 6853 of *LNCS*, Springer, Aug. 2011.