Procedia Computer Science

Volume 29, 2014, Pages 2219–2229

ICCS 2014. 14th International Conference on Computational Science

Image Noise Removal on Heterogeneous CPU-GPU Configurations

María G. Sánchez[1], Vicente Vidal[2], Josep Arnal[3], and Anna Vidal[4]

[1] Dpto. de Sistemas y Computación, Instituto Tecnológico de Cd. Guzmán, Cd. Guzmán, México
msanchez@dsic.upv.es

[2] Dpto. de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Valencia, Spain
vvidal@dsic.upv.es

[3] Dpto. de Ciencia de la Computación e Inteligencia Artificial, Universidad de Alicante, Alicante, Spain
arnal@ua.es

[4] Dpto. de Matemática Aplicada, Universidad Politécnica de Valencia, Valencia, Spain
avidal@mat.upv.es

Abstract

A parallel algorithm to remove impulsive noise in digital images using heterogeneous CPU/GPU computing is proposed. The parallel denoising algorithm is based on the peer group concept and uses an Euclidean metric. In order to identify the amount of pixels to be allocated in multi-core and GPUs, a performance analysis using large images is presented. A comparison of the parallel implementation in multi-core, GPUs and a combination of both is performed. Performance has been evaluated in terms of execution time and Megapixels/second. We present several optimization strategies especially effective for the multi-core environment, and demonstrate significant performance improvements. The main advantage of the proposed noise removal methodology is its computational speed, which enables efficient filtering of color images in real-time applications.

Keywords: parallel computing, noise removal in images, GPU, CUDA, multi-core, OpenMP

1 Introduction

Noise removal is an important problem in the field of image processing which has many applications in different fields. Very often noise corrupting the image is of impulsive nature. Impulsive noise is commonly caused by the malfunction of sensors and other hardware in the process of image formation, storage or transmission [2, 18]. This type of noise affects some individual pixels, changing their original values. The most usual model of impulsive noise is the Salt and Pepper noise or fixed value noise, which considers that the new, wrong, pixel value is an

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2014

doi:10.1016/j.procs.2014.05.207

extreme value within the signal range. This is the noise type we consider in this paper. Many algorithms have been proposed for correcting impulsive noise, for instance those mentioned in [3–5, 7–9, 14–18, 20]. In the context of color image filtering, one of the most used techniques is based on a vector ordering scheme defined through the ordering of aggregated distances [12]. Filters based on the ordering principle are efficient reducing the impulses, but they do not preserve fine image structures, which are treated as noise. In order to avoid the problems caused by the blurring properties of filters based on the ordering principle, a filtering method based on the concept of peer group was introduced in [6] and widely used in filtering design [3,4,8,10,18,19]. The peer group associated with the central pixel x_i is the set of neighboring pixels from the filtering window W beeing similar to x_i according to an appropriate metric value, this is, the nearest neighbors. This type of filters have recently shown good results in quality but they do not seem appropiate for real-time processing [3–6, 8, 9, 18, 19]. In this context, parallel computing emerges as a solution to decrease computational time. Nowadays parallel algorithms are widely present in noise removal literature [1, 13, 21]. In this paper, we introduce a parallel version of peer group based filters in order to retain their good quality results while trying to improve its performance, making them usable in real-time processing. We have tested these parallel algorithms developing programs for GPUs and multi-cores and we did an analysis of the best distribution of pixels in these two devices to take advantage of the hardware.

This paper is organized as follows: Section 2 explains the parallel noise removal method and how the parallel algorithm was implemented on GPUs and multi-core. Experimental results are shown in Section 3, and lastly, the conclusions are presented in Section 4.

2 Parallel image noise removal algorithm

Let the color image A be defined as a mapping $\mathbb{Z}^2 \to \mathbb{Z}^3$. 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 Ω. 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 $W = \{x_k \in \mathbb{Z}^2, k = 1, 2, \ldots, n\}$ represents a square filtering window (Figure 3) consisting of n color pixels centered at pixel x_1 (in the present study, $n = 3$ was considered). The parallel denoising algorithm introduced in this study uses the *peer group* of a central pixel x_i in a window W according to [18] and uses an *Euclidean metric*. In order to describe the parallel algorithm, and how the pixels were assigned to each computing element, we consider a domain decomposition of the image domain Ω in P subdomains $\{\Omega_i\}_{i=1}^{P}$, where P is the number of computing elements. Figure 1 shows an example of the image domain decomposition used in the experiments. Then, to detect and reduce the impulse noise the fuzzy peer group concept is used. For impulse noise reducing, we use the arithmetic mean filter AMF (e.g. [3]). Algorithm 1 shows the parallel filtering algorithm. The following lines detail the two steps of the filter. For a central pixel x_i in a $n \times n$ filtering window W and fixed the distance threshold $d \in [0, 1]$, we denote by $\mathcal{P}(x_i, d)$ the set

$$\mathcal{P}(x_i, d) = \{x_j \in W : \|x_i - x_j\|) \leq d\}. \tag{1}$$

Using the terminology employed in [3, 19], given a non-negative integer m, it will be called peer group of m peers associated to x_i a subset $\mathcal{P}(x_i, m, d)$ of $\mathcal{P}(x_i, d)$ formed by x_i and other m pixels, where $m \leq n^2 - 1$. Note that if $\mathcal{P}(x_i, d)$ contains $c + 1$ pixels then $\mathcal{P}(x_i, d)$ is a peer group of c peers. The algorithm performs two main steps. In the first step (*detection*) the pixels are labeled as either *corrupted* or *uncorrupted*. In the second step (*noise reduction*) corrupted pixels are corrected. Then, the detection and filtering steps are described for a single pixel x_i:

Algorithm 1 Parallel filtering algorithm.

Require: Image A, a domain decomposition $\{A_{\Omega_k}\}_{k=1}^{P}$, m, d
Ensure: Filtered image.

```
for k = 1, ..., P, in parallel do
  for x_i pixel in A_{Ω_k} do
    Impulse noise detection:
    Calculate P(x_i, d) :
              distance = ||x_i − x_j||
              if distance ≤ d then
                  pixel x_j ∈ P(x_i, d)
              endif
    if (#P(x_i, d) ≥ m + 1) then
      pixel x_i is free of impulse noise
    else
      Impulse noise reduction:
      x_i is an impulse and it is replaced with AMF_out
    end if
  end for
end for
```

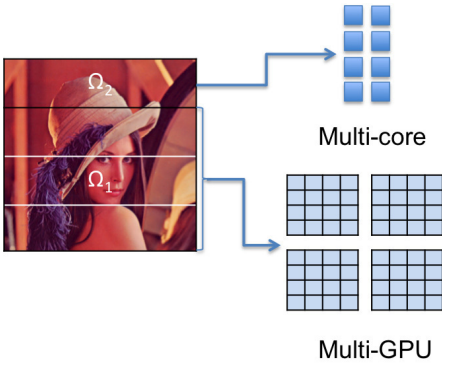


Figure 1: Distributed image: (3/4) on 4 GPUs and (1/4) on 8 cores.

- Detection: x_i is declared as corrupted if $\#\mathcal{P}(x_i, d) < (m+1)$, where m is the voting threshold and $\#\mathcal{P}$ the cardinality of set $\mathcal{P}$.
- Noise reduction: Given a pixel x_i previously marked as corrupted, it is replaced by the arithmetic mean of uncorrupted pixels in its window W. This is, the new value for $x_i(l)$ is $\frac{\sum_{x_j \in W'} x_j(l)}{\#W'}$, where W' is the set of uncorrupted pixels of W.

2.1 Comments on the GPU and multi-core implementation

We have developed three implementations. The first on multi-core using OpenMP, the second with CUDA on GPUs and the third is a combination of multi-core and GPUs. Figure 1 shows an example of the pixel distribution used in the experiments. The flowchart, Figure 2, shows the elimination of noise with these three implementations. In the detection process described in Algorithm 1, on the GPUs, the kernel was configured so that each thread processed one

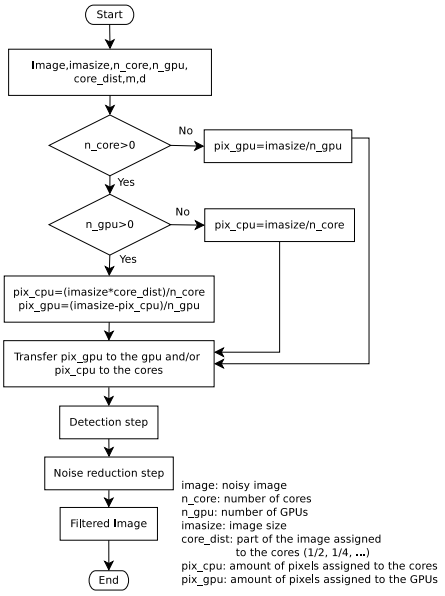


Figure 2: Flowchart describing the noise removal algorithm.

item of pixel data. The thread corresponding to the pixel x_i analyzes the $n \times n$ pixels of W, calculates the peer group and if this satisfies the cardinality $m+1$, the central pixel is diagnosed as uncorrupted; if not, it is diagnosed as corrupted. Given that AMF considers only uncorrupted pixels for mean computation, the noise reduction step cannot start until the detection phase is completed. In consequence, to ensure this synchronisation requirement, in the parallel implementation on GPU we have developed two kernels, so that the noise reduction kernel is not launched until the detecting kernel has finished. In multi-core two separate functions have been implemented. The arithmetic mean was used instead of the median for two main reasons: first, because in the calculation of the median, comparison operations are required and these operations are not recommended on the GPU. Second, because the computational cost of the arithmetic mean is lower than the median. In the GPU implementation, we reserve space in memory using 4 bytes per pixel and we access data through the texture memory. After several tests, this proved to be the best option.

2.2 Optimized detection step

To calculate the peer group and the Euclidean metric in the detection step, each pixel (i, j) in the domain Ω of the image forms a filtering window W with 3×3 pixels (see Figure 3). Euclidean metric considers the distance between the central pixel (i, j) and its 8 neighbors in the window W. Sequential algorithm performs $M \times N$ cycles, corresponding to the $M \times N$ image pixels. Figure 4 shows the execution of two cycles. Pixel (i, j) calculates the distance to its 8 neighbors. The value of the pixel (i, j) is used to calculate the distance from its 8 neighbors to it. In Figure 4 can be seen that when analyzing pixel $(3, 4)$, pixels $(2, 3), (2, 4), (2, 5), (3, 3)$ are accessed to calculate the distance between those pixels (Figure 2.2). These distances have been calculated previously, i.e., distance form pixel (3.4) to (2.3) is calculated when analyzing pixel $(2, 3)$. Then, in this process some distances are computed more than once. Concretely, distance

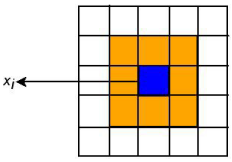
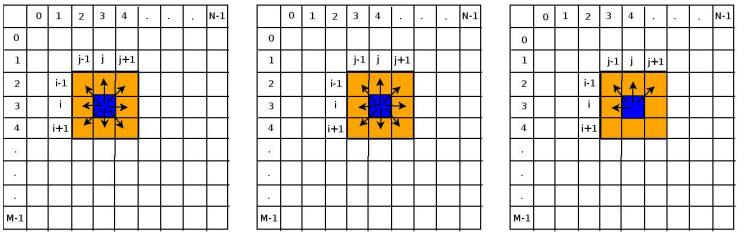


Figure 3: Filtering window W.



(a) Cycle i (b) Cycle $i+1$ (c) Memory access and repeated computations at cycle $i+1$

Figure 4: Detection step cycle.

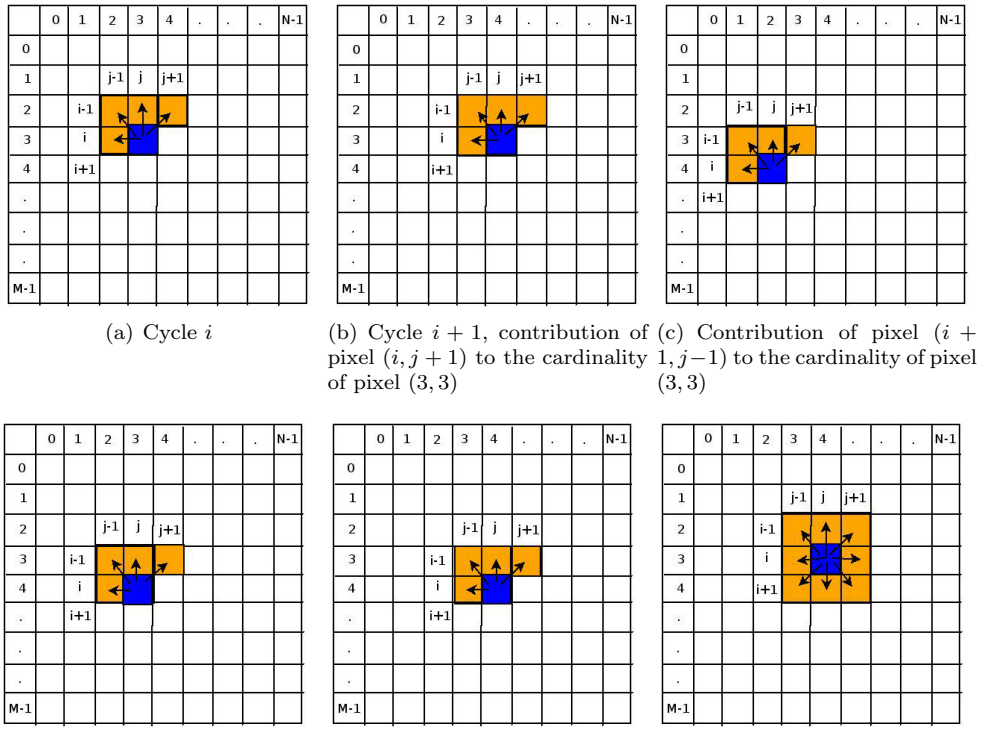
computation and data access in the peer group are repeated 4 times per pixel. In order to reduce the computation, 8 distances per pixel, we propose an optimized detection step, in which for each pixel (i, j) only distances to 4 neighbors, $(i-1, j-1), (i-1, j), (i-1, j+1), (i, j-1)$, are computed. Then to compute the cardinality of the peer group associated with the pixel (i, j), we use information computed when analyzing the neighboring pixels. This process can be seen in Figure 5. Then, pixel $(i-1, j-1)$ is classified as corrupt or not after pixel (i, j) is analyzed (see Figure 6).

3 Experimental results and discussion

We carried out specific experiments and developments using a Mac OS X Intel Xeon Quad-Core processor at 2 x 2.26 GHz with 8GB memory and with four NVIDIA GPUs (GeForce GT 120 with 512MB of memory (see [11])). This GPU supports CUDA Compute Capability 1.1. The CUDA toolkit 4.0.50 was used. Our implementation used C language and single-precision calculations. In order to adjust of the filter parameters d and m the filter performance has been analyzed in terms of PSNR as a function of d and m contaminating images with different densities of impulse noise. The best results were obtained when $d = 0.95$ and $m = 2$.

The results presented in this paper were obtained using the Lenna image (Figure 1) with RGB format and square dimensions 512×512, 1024×1024, 2048×2048 and 4096×4096 pixels. These images have been corrupted with 10% impulse noise.

To determine the number of threads per block that best fits the application, a heuristic study concluded that 64×64 threads per block gave lowest computational costs. Optimized detection step proposed in Section 2.2 has been implemented in parallel on multi-core, but not on GPUs. This is due to the fact that GPUs are particularly useful performing calculations, but they are penalized when memory access is needed. Moreover, optimized detection step algorithm generates many memory accesses conflicts among threads.



(a) Cycle i

(b) Cycle $i+1$, contribution of pixel $(i, j+1)$ to the cardinality of pixel $(3,3)$

(c) Contribution of pixel $(i+1, j-1)$ to the cardinality of pixel $(3,3)$

(d) Contribution of pixel $(i+1, j)$ to the cardinality of pixel $(3,3)$

(e) Contribution of pixel $(i+1, j+1)$ to the cardinality of pixel $(3,3$

(f) Distances and cardinality completed for pixel $(3,3)$

Figure 5: Optimized detection step.

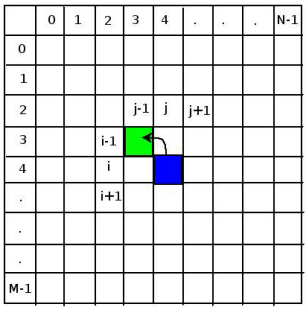


Figure 6: Pixel $(i-1, j-1)$ is classified as corrupt or not after pixel (i, j) is analyzed.

Table 1 shows the results obtained on the multi-core dividing the image only in cores. As it can be seen, the less time is presented when the image is divided among the 8 available cores, even for small size images (512×512 pixels). Next test consists of dividing image among the available GPUs. As shown in Table 2, the best results parallelizing an image smaller than 2048×2048 were obtained using 2 GPUs. When the image is larger than 2048×2048, the best time is obtained by using 4 GPUs. If times obtained on GPUs and CPUs are compared for

Table 1: Parallelizing Lenna image on multi-core. Time in seconds.

Image Size	1 core	2 cores	4 cores	6 cores	8 cores
512×512	0.020798	0.010955	0.005939	0.004343	**0.004316**
1024×1024	0.081733	0.041231	0.021845	0.015151	**0.014856**
2048×2048	0.323281	0.162790	0.082566	0.058701	**0.051113**
4096×4096	1.292198	0.647913	0,329343	0.229460	**0.185927**

each size, the best results were obtained using 8 cores for images smaller than 4096×4096, and for images of size 4096×4096 a better performance was obtained using 4 GPUs.

Table 2: Parallelizing Lenna image on GPUs. Time in seconds.

Image size	1 GPU	2 GPUs	4 GPUs
512×512	0.016530	**0.015097**	0.0224067
1024×1024	0.040734	**0.031762**	0.0349600
2048×2048	0.135072	0.091344	**0.0827400**
4096×4096	0.510320	0.319990	**0.1684350**

Table 3: Hybrid CPU-GPU. Image Size 4096×4096.

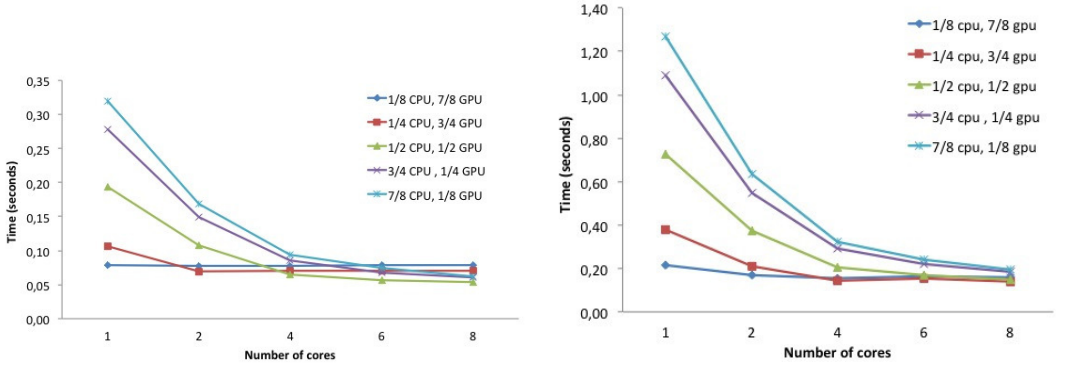
		Number of Cores				
Size on GPUs	Number of GPUs	1	2	4	6	8
7/8	1	0.432482	0.432538	0.432662	0.432729	0.432921
3/4	1	0.381904	0.371174	0.371137	0.371156	0.371461
1/2	1	0.724824	0.370513	0.261091	0.247571	0.24775
1/4	1	1.08635	0.54421	0.280308	0.198086	**0.1796**
1/8	1	1.268269	0.635293	0.324827	0.223815	0.21045
7/8	2	0.272231	0.272124	0.272006	0.272115	0.274267
3/4	2	0.382045	0.233651	0.233481	0.235535	0.234778
1/2	2	0.725656	0.370293	0.204804	**0.158565**	0.164457
1/4	2	1.086432	0.544539	0.277794	0.194978	0.180735
1/8	2	1.268063	0,635606	0.321942	0.224208	0.194173
7/8	4	0.214141	0.167618	0.155008	0.164813	0.160218
3/4	4	0.379341	0.209139	0.146158	0.153328	**0.139776**
1/2	4	0.729261	0.37423	0.202995	0.171603	0.147168
1/4	4	1.089055	0.547068	0.294627	0.221558	0.18351
1/8	4	1.267636	0.636421	0.324567	0.238853	0.19316

Table 3 presents the results obtained for image size 4096×4096 for different combinations of CPUs and GPUs. Table 3 also shows the portion of the image assigned to the GPUs. The rest was processed by the CPUs. Similar experiments were performed for the image size 2048×2048. For this size, Table 4 shows the best distribution using available hardware.

Figure 7 shows the results obtained when the image is divided into 4 GPUs and all the cores, for different image sizes. Figure 7 shows that the best results for the image 2048×2048 were obtained assigning 1/2 of the image in 8 cores and 1/2 in 4 GPUs. As can be seen in Figure 7, if the image size increases, then the best results were obtained assigning more processing on the GPU. As can be seen in the results, the parallelization performed using a combination of cores and GPUs gives better results than the parallelization performed only in 4 GPUs. For the image of 4096×4096 pixels, the reduction rate is 9.5% when the CPUs option is compared with the GPUs option, and 24.8% when CPUs model is compared with hybrid CPUs/GPUs model. Figure 8(a) presents time obtained using the optimized detection proposed in Section 2. Detection step is compared with the optimized detection step when the process is executed

Table 4: Hybrid CPU-GPU. Image size 2048 × 2048.

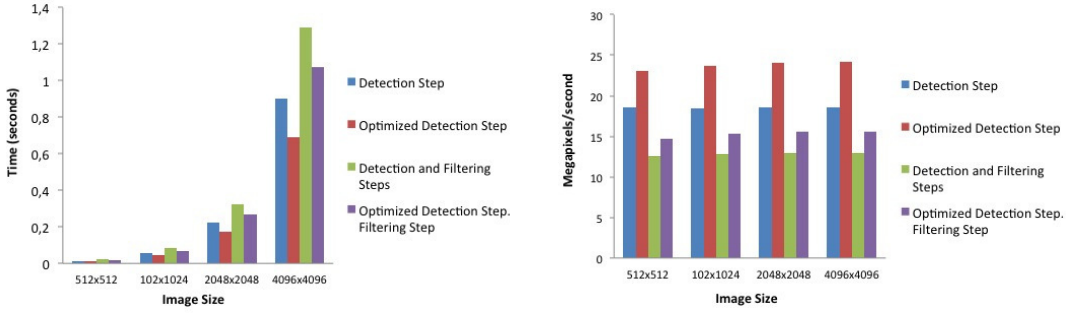
Number of GPUs	Optimal size on GPUs	Optimal number of CPUs	Time
1	1/8	8	0.051960
2	1/2	8	0.049696
4	3/4	8	0.053220



(a) Image size 2048 × 2048. (b) Image size 4096 × 4096.

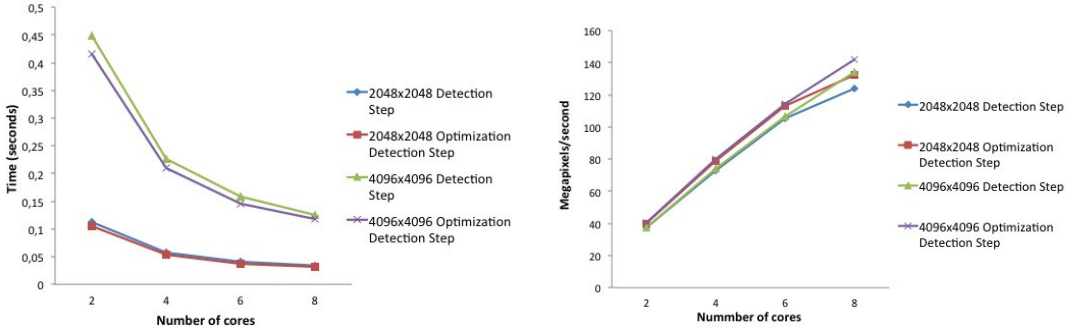
Figure 7: Hybrid CPU-GPU using 4 GPUs

sequentially in one CPU. It can be seen from Figure 8(a) that the improvement is significant. If the image size increases, improvement increases. Figure 8(b) compares Mpixels processed per second. For image of size 4096 × 4096 optimized version of detection step processes 6 Mpixels per second more than non optimized version. Figure 9 presents this comparison in parallel on multi-core. Figure 10 analyzes Gflops performed in the optimized detection step. In the sequential version, for the 4096 × 4096 image, the GFlops decrease from 1.501 to 1.08876 GFlops (27% reduction) when comparing detection step with optimized detection step. For the same image, in the parallel version using 8 cores, the GFlops decrease 41% in the optimized version.



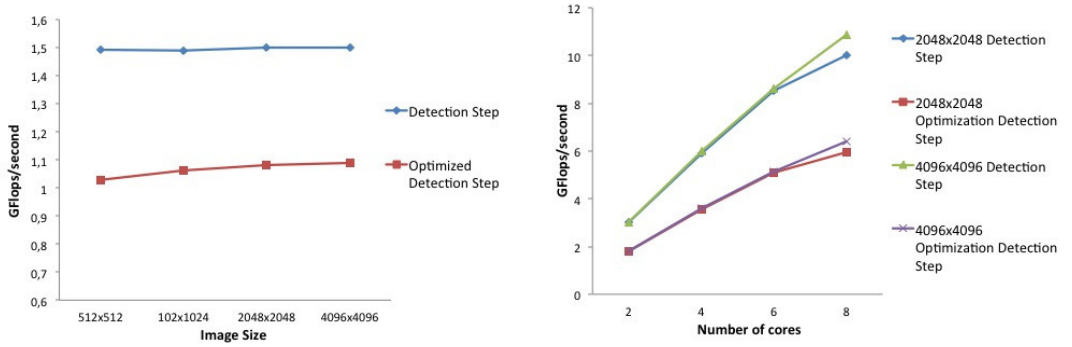
(a) Computational time (b) Megapixels per second

Figure 8: Optimized detection step. Sequential version.



(a) Computational time (b) Megapixels per second

Figure 9: Optimized detection step. Parallel version on multi-core.



(a) Sequential version (b) Parallel version on multi-core

Figure 10: GFlops performed in optimized detection step.

4 Conclusions

A parallel algorithm to remove impulsive noise of a digital image using heterogeneous CPU/GPU computing has been proposed. The denoising parallel algorithm is based on the peer group concept and uses an Euclidan metric. We have implemented it to be run on GPUs using the CUDA library and on multi-cores using OpenMP. This processing was divided into two steps: noise detection and noise elimination. For detection, the Euclidean metric and the concept of peer group were used. In the correction stage, corrupted pixel values were replaced by calculating the mean of those neighbors not labeled as corrupted in the detection step. Three implementations have been developed to be executed either on a multi-core, on several GPUs, or using a combination of CPUs and GPUs. Results showed that hybrid implementation CPU/GPU obtains the best performance. Several optimization strategies especially effective for the multi-core environment have been presented, demonstrating significant performance improvements. Numerical experiments show the efficient computational speed of the proposed noise removal methodology, which enables efficient filtering of color images in real-time applications.

5 Acknowledgements

This work was supported by the Spanish Ministry of Science and Innovation [grant number TIN2011-26254].

References

[1] Josep Arnal, Luis B. Sucar, Maria G. Sanchez, and Vicente Vidal. Parallel filter for mixed gaussian-impulse noise removal. In *Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA), 2013*, pages 236–241, Sept 2013.

[2] Charles Boncelet. Image noise models. In *A. C. Bovik (ed.), Handbook of Image and Video Processing*, pages 325–335, London, 2000. Academic Press.

[3] Joan-Gerard Camarena, Valentín Gregori, Samuel Morillas, and Almanzor Sapena. Fast detection and removal of impulsive noise using peer groups and fuzzy metrics. *Journal of Visual Communication and Image Representation*, 19(1):20–29, 2008.

[4] Joan-Gerard Camarena, Valentín Gregori, Samuel Morillas, and Almanzor Sapena. Some improvements for image filtering using peer group techniques. *Image Vision Comput.*, 28(1):188–201, 2010.

[5] Joan-Gerard Camarena, Valentín Gregori, Samuel Morillas, and Almanzor Sapena. Two-step fuzzy logic-based method for impulse noise detection in colour images. *Pattern Recognition Letters*, 31(13):1842–1849, 2010.

[6] C. Kenney, Y. Deng, B. S. Manjunath, and G. Hewer. Peer group image enhancement. *Trans. Img. Proc.*, 10(2):326–334, February 2001.

[7] Tom Melange, Mike Nachtegael, and Etienne E. Kerre. Fuzzy random impulse noise removal from color image sequences. *Trans. Img. Proc.*, 20(4):959–970, April 2011.

[8] Samuel Morillas, Valentín Gregori, and Antonio Hervás. Fuzzy peer groups for reducing mixed gaussian-impulse noise from color images. *IEEE Transactions on Image Processing*, 18(7):1452–1466, 2009.

[9] Samuel Morillas, Valentín Gregori, and Guillermo Peris-Fajarnés. Isolating impulsive noise pixels in color images by peer group techniques. *Computer Vision and Image Understanding*, 110(1):102–116, 2008.

[10] Samuel Morillas, Valentn Gregori, Guillermo Peris-Fajarnés, and Almanzor Sapena. Local self-adaptive fuzzy filter for impulsive noise removal in color images. *Signal Processing*, 88(2):390–398, 2008.

[11] NVIDIA Corporation. NVIDIA GeForce GT 120 Graphic Card, 2012. `http://www.geforce.com/hardware/desktop-gpus/geforce-gt-120`.

[12] Konstantinos N. Plataniotis and Anastasios N. Venetsanopoulos. *Color image processing and applications*. Springer-Verlag New York, Inc., New York, NY, USA, 2000.

[13] Maria G. Sánchez, Vicente Vidal, Jordi Bataller, and Josep Arnal. A parallel method for impulsive image noise removal on hybrid CPU/GPU systems. *Procedia Computer Science*, 18(0):2504 – 2507, 2013. 2013 International Conference on Computational Science.

[14] Stefan Schulte, Samuel Morillas, Valentín Gregori, and Etienne E. Kerre. A new fuzzy color correlated impulse noise reduction method. *IEEE Transactions on Image Processing*, 16(10):2565–2575, 2007.

[15] Stefan Schulte, Mike Nachtegael, Valérie De Witte, Dietrich Van der Weken, and Etienne E. Kerre. A fuzzy impulse noise detection and reduction method. *IEEE Transactions on Image Processing*, 15(5):1153–1162, 2006.

[16] Stefan Schulte, Valérie De Witte, Mike Nachtegael, Dietrich Van der Weken, and Etienne E. Kerre. Fuzzy two-step filter for impulse noise reduction from color images. *IEEE Transactions on Image Processing*, 15(11):3567–3578, 2006.

[17] Stefan Schulte, Valérie De Witte, Mike Nachtegael, Dietrich Van der Weken, and Etienne E. Kerre. Fuzzy random impulse noise reduction method. *Fuzzy Sets and Systems*, 158(3):270–283, 2007.

[18] Bogdan Smolka. Peer group switching filter for impulse noise reduction in color images. *Pattern Recognition Letters*, 31(6):484–495, 2010.

[19] Bogdan Smolka and Andrzej Chydzinski. Fast detection and impulsive noise removal in color images. *Real-Time Imaging*, 11(5-6):389–402, 2005.

[20] Abdullah Toprak and Inan Güler. Impulse noise reduction in medical images with the use of switch mode fuzzy adaptive median filter. *Digital Signal Processing*, 17(4):711–723, 2007.

[21] Zhiyi Yang, Yating Zhu, and Yong Pu. Parallel image processing based on CUDA. In *Proceedings of the International Conference on Computer Science and Software Engineering, CSSE 2008, Volume 3: Grid Computing / Distributed and Parallel Computing / Information Security, December 12-14, 2008, Wuhan, China*, pages 198–201. IEEE Computer Society, 2008.