Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 2504 – 2507

Procedia Computer Science

International Conference on Computational Science, ICCS 2013

A Parallel Method for Impulsive Image Noise Removal on Hybrid CPU/GPU Systems

M.G. Sánchez[a,b], V. Vidal[b], J. Bataller[b], J. Arnal[c,*]

[a]*Departamento de Sistemas y Computación, Instituto Tecnológico de Cd. Guzmán, Cd. Guzmán, 49100, Jal. México*
[b]*Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Grao de Gandia Valencia, 46730, Spain*
[c]*Departamento de Ciencia de la Computación e Inteligencia Artificial, Universidad de Alicante, Alicante, 03071, Spain*

Abstract

A parallel algorithm for image noise removal is proposed. The algorithm is based on peer group concept and uses a fuzzy metric. An optimization study on the use of the CUDA platform to remove impulsive noise using this algorithm is presented. Moreover, an implementation of the algorithm on multi-core platforms using OpenMP is presented. Performance is evaluated in terms of execution time and a comparison of the implementation parallelised in multi-core, GPUs and the combination of both is conducted. A performance analysis with large images is conducted in order to identify the amount of pixels to allocate in the CPU and GPU. The observed time shows that both devices must have work to do, leaving the most to the GPU. Results show that parallel implementations of denoising filters on GPUs and multi-cores are very advisable, and they open the door to use such algorithms for real-time processing.

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

1. Introduction

Impulsive noise is commonly caused by the sensor malfunction and other hardware in the process of image formation, storage or transmission [1]. 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.

In recent years, the incorporation of GPUs (Graphics Processing Units) in graphic cards has achieved significant improvements in computational speed, offering a high parallel processing level. Due to this, developments based on this hardware have become increasingly widespread, not only for graphic implementations but also for general purpose applications. The most commonly used programming platform for these graphic cards is CUDA (Compute Unified Device Architecture) [2]. Whilst it is relatively easy to use the CUDA platform to program the GPU, the problem lies in the difficult task of optimizing application performance, due to several hardware restrictions and the multiple types of memories included. Thus, it is necessary to carry out a specific study in order to identify the best approach for using the resources offered by CUDA in each case. Such a study is one of the subjects of the present paper, applied in this case to noise removal in images. In the present study, we

*Corresponding author. Tel.: +34-96-590-3900 ; fax: +34-96-590-3902 .
E-mail address: arnal@ua.es.

Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science
doi:10.1016/j.procs.2013.05.429

implemented a parallel algorithm for image noise removal based on the family of algorithms proposed in [3, 4, 5]. These methods are based on peer group concept and use a fuzzy metric. They have been proved to be very fast and achieve a good trade-off between noise suppression and signal-detail preservation. Due to this in this paper we present a parallel method based on them, but adapted to obtain a good real-time performance on GPUs and on hybrid CPU/GPU architectures. The noise removal process was divided into two steps: erroneous pixel detection, and the elimination of these pixels. We analyse the access time to global memory and use the texture memory in order to reduce it. We have tested this parallel algorithm on GPUs and multi-cores and we did an analysis of the best pixel distribution in these devices to take advantage of the hardware.

This paper is organised as follows: Section 2 explains the proposed parallel noise removal method. Section 3 discusses how the parallel algorithm was implemented for the GPUs and multi-core. Experimental results are shown in Section 4, and finally, the conclusions are presented in Section 5.

2. Parallel Image Noise Removal Algorithm

The parallel denoising algorithm introduced in this study uses the *peer group* of a central pixel x_i in a window W according to [1] but using a fuzzy metric instead. The fuzzy distance between the vectors x_i and x_j of the colour image is given by the following function:

$$M(x_i, x_j) = \prod_{l=1}^{3} \frac{min\left\{x_i(l), x_j(l)\right\} + k}{max\left\{x_i(l), x_j(l)\right\} + k}. \tag{1}$$

where (x_i (1), x_i (2), x_i (3)) is the colour vector for the pixel x_i in RGB (Red, Green and Blue) format and x_j is a neighbour pixel of x_i. In [6], was shown that $k = 1024$ is an appropriate setting to maintain the image quality, and this was therefore the value used in the present study. 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

$$\mathcal{P}(x_i, d) = \{x_j \in W : M(x_i, x_j) \geq d\}. \tag{2}$$

Using the terminology employed in [4, 7], 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 $m \leq n^2 - 1$. The algorithm performs two main steps. In the first step (*detection*), described in Algorithm 1, the pixels are labelled as either *corrupted* or *uncorrupted*. In the second step (*filtering*), presented in Algorithm 2, corrupted pixels are corrected. In the detection process, the kernel was configured so that each thread processed one item of pixel data. The thread corresponding to the pixel x_i analyses the $n \times n$ pixels of W (in the present study, $n = 3$ was considered), calculates the peer group and if this satisfies the cardinality $m + 1$, i.e. $\#\mathcal{P}(x_i, d) \geq m + 1$, the central pixel is diagnosed as uncorrupted; if not, it is diagnosed as corrupted. In the filtering process, the well-known arithmetic mean filter (AMF) (e.g. [4, 8]) is used.

Algorithm 1 Detection of erroneous pixels.

Require: m, d, W.
Ensure: The image with the pixels classified as corrupt or not.

```
Calculate P(x_i, d) :
    Each thread calculates:
        distance = M(x_i, x_j)
        if distance ≥ d then
            pixel x_j ∈ P(x_i, d)
        endif
if (#P(x_i, d) ≥ m + 1) then
    x_i is declared as non-corrupted pixel
else
    x_i is declared as corrupted pixel
end if
```

Algorithm 2 Elimination of erroneous pixels.

Require: n, threads corresponding to the pixels labelled as corrupted.
Ensure: Filtered Image.
1: Each thread builds its window W of pixels.
2: Calculate the AMF of uncorrupted pixels from W.
3: Thread corresponding to the pixel x_i replaces value obtained by the AMF.

3. Comments on the GPU and multi-core implementation

We decomposed the algorithm into two phases –detection and filtering– not only to follow the separation of concerns principle, but also because we use the AMF for replacing corrupted pixels. Given that AMF considers only uncorrupted pixels for mean computation, the filter phase cannot start until the detection phase is done. In consequence, to ensure this synchronisation requirement, in our parallel implementation on GPU we have developed two kernels, so that the filtering kernel is not launched until the detecting kernel has finished. In multi-core two separate functions have been implemented. In the implementation on GPU, we use texture to access data from GPU. We have developed three implementations. The first is in multi-core using OpenMP, the second with CUDA on GPUs. The third is a combination of multi-core and GPUs. Figure 1 shows the distribution of an image using cores and GPUs.

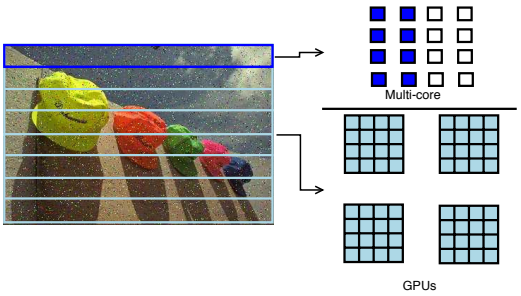

Fig. 1. Distributed image on 4 GPUs and 8 cores.

4. Experimental Results

The CPU used was a Mac OS X Intel Xeon Quad-Core processor at 2×2.26 GHz with 8GB memory and with four NVIDIA GPUs. Each GPU is a GeForce GT 120 [9]. This card has 512MB of memory, 4 multiprocessors, 32 CUDA cores with processor clock 1400 MHz, 16KB shared memory and 16GB/s memory bandwidth. 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. The results were obtained using the Caps image (Figure 1) from the Kodak image database [10]. MATLAB command *imnoise* was used to generate the noisy image.

Figure 2 shows the speedup when comparing the sequential version running on CPU and the parallel version on multi-core and GPUs. On the multi-core the speedup increases with the number of used cores for all image sizes. On the GPUs, parallelising an image smaller than 1536×1024 is best done in one GPU. Dividing the image into 4 GPUs smaller than 3072×2048, leads to a greater time than dividing the distribution on two GPUs. We observe that when the image is large, the optimal distribution is obtained by using four GPUs, but not for small images (less than 3072×1048). Figure 3 shows the results obtained when the image is divided into 4 GPUs and all the cores. The best time is shown when more load is assigned to the GPU leaving less on the cores.

5. Conclusions

A parallel algorithm based on peer group concept that uses a fuzzy metric has been proposed to detect and remove impulsive noise in digital images. We have implemented it to be run on GPUs using CUDA and on multi-cores using OpenMP. We conducted experiments to adjust our algorithm, and to compare it with the sequential

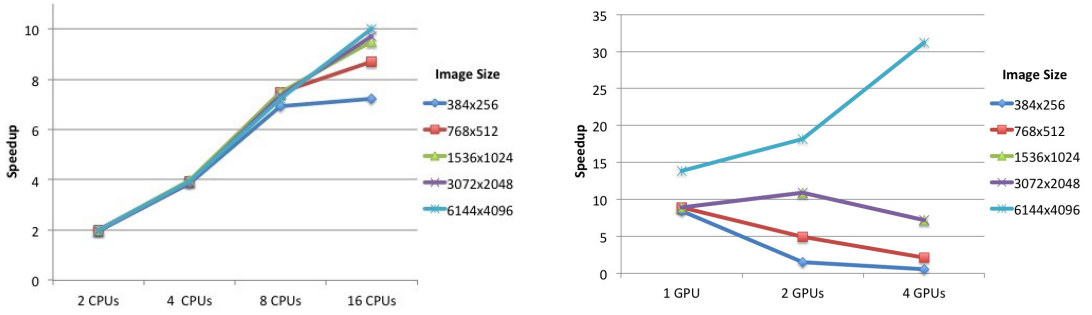

Fig. 2. Speedup for different image sizes. Caps image.

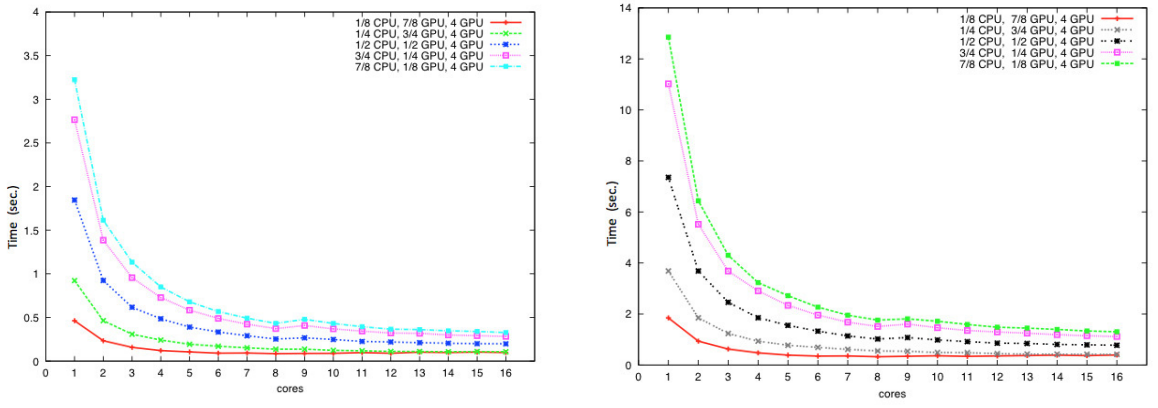

Fig. 3. (a) Size 3072 × 2048.; (b) Size 6144 × 4096.

version of the algorithm obtaining outstanding results in speedup. It was concluded that parallel implementations of denoising filters on GPUs and multi-cores are very advisable, and they open the door to use such algorithms for real-time processing. For future works, we plan to implement this algorithm to detect edges in an image.

Acknowledgements

This work was supported by the Spanish Ministry of Science and Innovation [Project TIN2011-26254]. M. G. Sánchez would like to acknowledge DGEST ITCG for the scholarship awarded through the PROMEP program.

References

[1] B. Smolka, Peer group switching filter for impulse noise reduction in color images, Pattern Recognit. Lett. 31 (6) (2010) 484–495.
[2] NVIDIA (2012) http://www.nvidia.com/page/home.html.
[3] S. Morillas, V. Gregori, G. Peris-Fajarnés, Isolating impulsive noise pixels in color images by peer group techniques, Comput. Vision Image Understanding 110 (1) (2008) 102–116.
[4] J.-G. Camarena, V. Gregori, S. Morillas, A. Sapena, Fast detection and removal of impulsive noise using peer groups and fuzzy metrics, J. Visual Commun. Image Represent. 19 (1) (2008) 20 – 29.
[5] J.-G. Camarena, V. Gregori, S. Morillas, A. Sapena, Some improvements for image filtering using peer group techniques, Image Vision Comput. 28 (1) (2010) 188–201.
[6] S. Morillas, V. Gregori, G. Peris-Fajarnés, P. Latorre, A fast impulsive noise color image filter using fuzzy metrics, Real-Time Imaging 11 (5-6) (2005) 417–428.
[7] B. Smolka, A. Chydzinski, Fast detection and impulsive noise removal in color images, Real-Time Imaging 11 (5-6) (2005) 389–402.
[8] A. Toprak, I. Güler, Impulse noise reduction in medical images with the use of switch mode fuzzy adaptive median filter, Digital Signal Process. 17 (4) (2007) 711–723.
[9] NVIDIA GeForce GT 120 Graphic Card (2012), http://www.geforce.com/hardware/desktop-gpus/geforce-gt-120.
[10] Kodak Lossless True Color Image Suite (2012), http://r0k.us/graphics/kodak/index.html.