DNLM-MA-P: A Parallelization of the Deceived Non Local Means Filter with Moving Average and Symmetric Weighting

Manuel Zumbado[1], Jorge Castro[2], Saúl Calderón[3]
[1,3]PAttern Recognition and MAchine Learning Group (PARMA-Group)
School of Computing, Instituto Tecnológico de Costa Rica
[2]Advanced Computing Laboratory
Costa Rica National High Technology Center (CeNAT)
[1]manzumbado@ic-itcr.ac.cr, [2]jcastro@cenat.ac.cr, [3]sacalderon@itcr.ac.cr

***Abstract*—In this paper we present a novel computational optimization of the deceived non local means filter using moving average and symmetric weighting. We compare the proposed optimization with different approaches that reduce the computational cost of the deceived non local means filter. Furthermore, we also assess the impact of parallelizing the different optimization approaches, evaluating the execution time and scalability in a many-core architecture. The proposed approach for the non-parallel implementation achieved a 90x speedup, while its parallelized implementation yielded a speedup of up to 1662x.**

***Index Terms*—Non Local Means filter, Image Processing, Parallelization, Optimization, Xeon Phi Knights Landing.**

I. Introduction

Image and signal denoising refers to suppressing partially or completely irrelevant energy from a signal, as images are affected by many noise sources like sensor sensibility or imperfections in the capture and communication processes. It is a well addressed problem in literature, with vast number of techniques proposed [1]. Noisy input images decrease the performance of image processing algorithms such as segmentation, object tracking and classification [2]–[4]. Thus, it becomes an important issue to incorporate denoising and enhancement techniques as a preprocessing stage, to increase performance of latter steps. Furthermore, the usage of a preprocessing stage has improved the model precision of popular Convolutional Neural Network (CNN) based approaches for image processing related tasks [5], [6]. CNN models build feature extractors upon training data, in spite of classical hand-crafted feature extraction approaches, which leads to the intuition that CNN methods are more robust to noisy training data than hand-crafted approaches. However, previously reported results [5], [6] suggest that CNN based methods are also significantly sensitive to noisy data, which leads to the need of implementing preprocessing approaches for data denoising and relevant information enhancement.

Nonetheless, implementing complex and powerful image preprocessing approaches for enhancing the overall precision of different image based pattern recognition tasks comes at the price of increasing the need of computational resources. This encourages the implementation of increasingly efficient computational optimizations of popular preprocessing techniques.

In this paper, we implement and compare different optimizations of the Deceived Non Local Means (DNLM) filter [7], which performs both image enhancement and denoising, using the popular edge preserving Non Local Means (NLM) kernel designed for image denoising [8].

The DNLM brute force implementation presents a high computational cost, which has been addressed previously [9], but still is not suitable for preprocessing large data sets. For example in the case of the automatic segmentation of cell activity videos, researchers need to analyze hundreds of thousands of images [2]. Also, preprocessing large datasets for CNN models is useful as recent work reveals [6]. For those reasons, in this paper we focus on techniques to reduce the DNLM computational cost, and moreover, we explore its parallelization to further reduce its execution time. We evaluate the parallelization of the original brute force DNLM implementation and two computational optimizations of the filter on Xeon Phi Knights Landing (KNL) architecture. Both optimizations of the DNLM filter and its original implementation are parallelized using task level parallelism (OpenMP) and data level parallelism (vectorization). We measured the speedup of the parallelized filters against its sequential version.

The paper is organized as follows. First, in Section II we describe the DNLM filter and previous optimization

and parallelization approaches for the NLM filter. Then, we present the proposed parallelization layout for the DNLM filter and the technology used from both software and hardware perspective in Section III. The performed experiments and its results are detailed in Section IV. Finally, we present the conclusions and future work in Section V.

II. Background

A. DNLM Description

As mentioned earlier, the DNLM performs both image denoising and the enhancement of contrast and edges. The DNLM is a particularization of the Deceived Weighted Averaging Framework (DeWAFF). The DNLM uses the NLM kernel to filter an image $F_{\mathbf{USM}}$ enhanced with any Unsharped Masking (USM) technique, but using the original image U for the kernel weighting. This is done to avoid the often resulting ringing artifacts in USM approaches. For instance, image $F_{\mathbf{USM}}$ can be calculated as a simple linear combination $F_{\text{USM}} = U + \lambda L$, with L the negative laplacian approximation, and λ the sharpening gain.

The DNLM filter is formally expressed in (1), where p is a specific pixel and Ω the current search window of $|\Omega| = S$ pixels around pixel p:

$$Y(p) = \frac{\left(\sum_{m\in\Omega} \psi_{\text{NLM}}\left(U,p,m\right) F_{\text{USM}}\right)}{\left(\sum_{m\in\Omega}\ \psi_{\text{NLM}}\left(U,p,m\right)\right)} \quad . \tag{1}$$

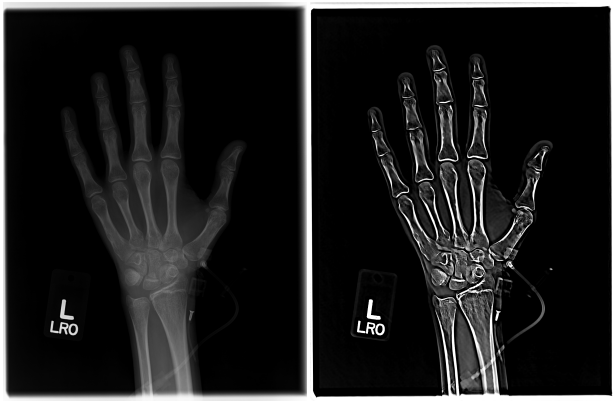
The DNLM uses the non local means weighting function [8], which measures per pixel neighborhood dissimilarity. The weighting function of the NLM filter is defined in (2):

$$\psi_{\text{NLM}}(U,p,m) = \exp\left(-\frac{\|\ \overrightarrow{\eta}(\omega,m) - \overrightarrow{\eta}(\omega,p)\|^2}{h}\right), \tag{2}$$

with $\overrightarrow{\eta}(\omega,m)$ the vectorized neighborhood of $|\omega| = W$ pixels for the pixel m within U, and h controls filter smoothing. The NLM kernel showed superior denoising and enhancement results when compared with other kernels as the bilateral filter [2].

B. Optimization approaches for the NLM and the DNLM

Many sequential optimizations have been proposed to reduce the time complexity of the NLM filter which, for its brute force implementation, is of $\mathcal{O}(N \cdot S \cdot W)$ for gray-scale images, where N, S, and W are the number of pixels in the image, search window, and neighborhood window, respectively. Some of these optimizations are based on approximations of the NLM filter [10] or modifications that improve its results [11]–[16], however we discarded both categories in this paper, as we aim to first preserve similar behavior of the DNLM filter.



(a) Original image (b) DNLM-MA output

Figure 1. Output of the DNLM filter with a window size of 21×21, a neighborhood size of 7×7, $h = 5$, and $\lambda = 15$. The input sample is an X-Ray image for an automatic patient age assessment approach based on CNNs with DNLM filter as a preprocessing stage [6].

In this section we focus on exploring computational optimizations of the NLM filter, to port them to the DNLM filter. One technique within this category uses integral images and the Fast Fourier Transform (FFT) to compute the distance between different neighborhoods, reducing the time complexity of the filter to $\mathcal{O}(N \cdot S)$ [17], [18]. The distance computation has been also optimized exploiting the weight symmetry between pixels, using a moving average, and reordering the filter loops, reducing the computation time to $\mathcal{O}(N \cdot S)$ [19]. Later, a similar approach was proposed using a separable convolution to implement the moving average filter [20], leading to a similar time complexity reduction. Also, an optimization based on the Sum of Invariant Lines (SIL) yields a time complexity of $\mathcal{O}(N \cdot S \cdot \sqrt{W})$ [21].

Since the DNLM filter requires the decoupling of the weighted image and the filtered image we focus on the optimizations that meet that requirement and best reduce the time complexity of the algorithm. Those optimizations are the proposed by Wang et al. [17] and Condat [20]. The optimization proposed by Goossens et al. [19] is very similar to the proposed by Condat, but the latter is more suitable to parallelize since it implements the moving average as a separable convolution, removing data dependencies and leading to a computational complexity of $\mathcal{O}(N \cdot S \cdot \sqrt{W})$. The optimization proposed by Wang has already been partially implemented without symmetry assumption for the DNLM filter, achieving a speedup of up to ten times [9]. In this work we implement the optimization proposed by Condat for the NLM to optimize the DNLM.

C. Previous parallelization approaches for the non local means filter

A summary of the proposed parallelization methods found in literature is shown in Table I. The first parallelization of the filter was done using multiple threads to process 3D medical images in a server with eight Xeon processors [22]. Later, a Single Instruction Multiple Data (SIMD) architecture was used to parallelize the filter in a server with eight Dual-Core AMD Opteron processors [23]. Also, GPU implementations of the filter have been proposed based on Goossens' optimization on DirectX [24] and Condat's optimization on CUDA platform [25], [26], where the first and the latter work reported higher speedup than previous CPU implementations. A slightly higher speedup to the one obtained by Marques and Pardo [24] was achieved parallelizing the filter with MPI in a distributed memory system with 1024 cores [27]. A more complex implementation of an improved NLM filter on hybrid parallel architectures, combined MPI with P-threads and MPI with CUDA to achieve a high speedup [28]. Finally, two implementations of the filter on Intel Xeon Phi platforms were proposed using OpenMP [29], [30] and OpenCL [29].

Although the speedup obtained on GPU platforms is usually higher than the ones obtained on Intel Xeon Phi platforms, the latter platform provides a more flexible and simple programming style that is very suitable for developing parallel image processing algorithms prototypes, since it uses the x86 instruction set [30].

Table I
PREVIOUS PARALLEL IMPLEMENTATION METHODS FOR THE NLM FILTER. ONLY THE MAXIMUM SPEEDUP WITH RESPECT TO THE SEQUENTIAL-UNOPTIMIZED VERSION OF THE FILTER IS SHOWN.

Implementation	Architecture	Parallel tool	Speedup
Coupe et al. [22]	8 Intel Xeon CPU	Threading library	50×
Darbon et al. [23]	8 Dual-Core AMD Opteron CPU	Vectorization instructions	110×
Mingliang et al. [25]	NVIDIA Quadro FX 480	CUDA	40×
Gossens et al. [26]	NVIDIA GeForce 9600 GT	DirectX	402×
Marques and Pardo. [24]	NVIDIA GeForce GTX 680	CUDA	718×
Shi et al. [27]	1024 cores of SuperMUC	MPI	740×
Nguyen et al. [28]	8 Intel Xeon CPU	MPI and PThreads	148×
Nguyen et al. [28]	8 NVIDIA Tesla C2050	MPI and CUDA	510×
Zhu et al. [29]	Intel Xeon Phi 7110P	OpenMP	87×
Zhu et al. [29]	Intel Xeon Phi 7110P	OpenCL	108×
Huang et al. [30]	Intel Xeon Phi	OpenMP	32×

III. PROPOSED METHOD: ACCELERATING THE DNLM

In this work, we propose and develop both parallel and vectorized versions of two computational optimizations for the DNLM filter, namely the DNLM-IFFT and the DNLM-MA. Moreover, the proposed implementations take advantage of the Xeon Phi KNL platform features. Vectorization is successfully achieved by using Intel Performance Primitives (IPP) with AVX-512 support for almost all pixel operations.

The optimized DNLM-MA filter is based on the Moving Average (MA) and symmetry approach proposed by Condat [20]. Following the formal definition of the DNLM filter, a pixel y can be expressed in terms of the displacement from pixel x as $y = x + \Delta$, within a search window of s pixels. For instance, for a search window of 5×5, with $s = 25$ pixels, Δ can take the values $\Delta = 0$, $\Delta = 1$, $\Delta = 2$, and in general goes up to $\Delta = S/2$. Thus, the weighting function of NLM accomplishes symmetry as:

$$\psi_{\text{NLM}}(U, x, x + \Delta) = \psi_{\text{NLM}}(U, y, y - \Delta) \ . \quad (3)$$

The weights are computed by swapping the loops, as it iterates for each displacement Δ, instead of iterating over all of the pixels U. For each iteration, the squared difference is computed for the entire image as:

$$D_{\Delta}(x) = \left(U(x) - U(x + \Delta)\right)^{\circ 2} \ , \quad (4)$$

where the squared power is a pixel wise operation following the Hadamard notation.

Latter, a convolution of the result D_{Δ} with a moving average kernel of size W is performed by:

$$E_{\Delta}(x) = D_{\Delta}(x) * G \ , \quad (5)$$

with G an unary matrix with the neighborhood dimensions.

Thus, the weight function of the DNLM-MA is given by the element-wise exponential function, also written with the Hadamard notation as:

$$\psi_{\Delta} = \exp^{\circ}\left(-\frac{E_{\Delta}(x)}{h}\right) \ . \quad (6)$$

The DNLM-MA filter response for each iteration is given by the pixel wise product of the weights in ψ_{Δ} and F_{USM} as:

$$Y_{\Delta} = \psi_{\Delta} \circ F_{\text{USM}} \ . \quad (7)$$

To calculate the final filter response $Y_{\text{DNLM-MA}}$, we calculate first the normalization coefficient c by accu-

mulating the weights for all the displacements Δ in matrix $\psi_{\text{DNLM-MA}}$ as:

$$\psi_{\text{DNLM-MA}} = \sum_{\Delta=0}^{q} \psi_{\Delta} \ , \tag{8}$$

and latter sum up all its elements to reach $c = \Sigma_i \psi_{\text{DNLM-MA}}(i)$. To reach the filter output, partial filter responses calculated in (7) are accumulated, normalized by c, as:

$$Y_{\text{DNLM-MA}} = \sum_{i=0}^{q} c^{-1} Y_{\Delta} \ . \tag{9}$$

The parallelized DNLM-MA is done with the concurrent computation of each Δ iteration. However, since the search window size S is not large enough to provide enough parallelism for outer loop parallelization, we opt to parallelize the computation of (4) to (7) to take advantage of the many-core architecture.

Is important to highlight that the weight computation in the DNLM-MA performs operations over all the image for each iteration, corresponding to each displacement Δ, in spite to the DNLM-IIFFT and brute f force implementations. This leads us to the intuition that DNLM-MA filter has higher memory bandwidth requirements than the DNLM-IIFFT and DNLM filters. Thus, the scaling of the DNLM-MA version might not be as good as the other DNLM implementations.

IV. Evaluation

A. Experimental setup

All the tests in this work were performed in an Intel Xeon Phi platform which includes architectural optimization features suitable for high performance computing purposes. One of its main highlights is the incorporation of a 16GB on-chip high bandwidth memory, which allows several memory layout configurations, providing enough bandwidth and good timing to help hide the main memory latency [31]. It also features 64 cores with simultaneous multi threading up to 256 threads, organized in a mesh grid of 32 tiles. Each tile comprises two cores and 1MByte of shared L2 cache. Each core contains two vector processing units with latest AVX-512 instruction set support, allowing a peak computation of 64 single precision or 32 double precision operations per cycle [31]. To perform the experiments we used a default gray scale input of 1024×1024 pixels, a search window size of $S = 21 \times 21$, and a neighborhood window size of $W = 7 \times 7$. All the reported results correspond to the average of 10 executions or replicas for each evaluated program. To execute each parallel implementation of the filter, we choose as default the number of threads that minimizes its runtime using the default parameters values for S, W, and image size N. Thus, DNLM-P was executed using 256 threads, the DNLM-IIFFT-P with 128 threads, and DNLM-MA-P with 32 threads. It is important to highlight that all parallel and sequential implementations of the DNLM filter used IPP directives that were automatically vectorized for KNL architecture.

B. Experiments and Results

Table II shows a comparison of the runtime for each implemented version of the DNLM filter. It reveals that the DNLM-MA outperforms by $88.30\times$ the DNLM-IFFT implementation. The coefficient of variation for all measured runtime is equal or less than 0.06. Also, the DNLM-MA-P implementation is the fastest parallelization, achieving a $665.63\times$ speedup over the unoptimized sequential DNLM filter. However, the DNLM-IIFFT-P and DNLM-P obtain speedups higher than $100\times$ compared to the unoptimized sequential DNLM filter. Table II also reveals that the DNLM-MA-P obtains a lower speedup ($7\times$) than the DNLM-IFFT-P ($77\times$) and DNLM-P ($109\times$) with respect to its sequential version.

Table II
AVERAGE RUNTIME AND SPEEDUP OF ALL THE DNLM FILTER IMPLEMENTATIONS.

Filter	Duration [s]	Coef. of var.	Speedup [×]
DNLM	163.71 ± 1.32	0.01	1
DNLM-IIFFT	76.68 ± 0.06	0.00	2.14
DNLM-MA	1.81 ± 0.01	0.00	90.44
DNLM-P	1.5 ± 0.10	0.06	108.99
DNLM-IIFFT-P	1.00 ± 0.00	0.00	163.92
DNLM-MA-P	$\mathbf{0.25 \pm 0.00}$	**0.01**	**665.63**

The effect of changing the image size in the overall speedup of the DNLM-MA-P implementation is displayed in Table III. It shows that the speedup of DNLM-MA-P monotonically increases, going from $206.95\times$ to $759.42\times$ for image sizes going from 256×256 to 4096×4096 pixels. However, the increment in the speedup becomes smaller and smaller for images bigger than 1024×1024 pixels, and for images of 8192×8192, the speedup actually decreases to $318.38\times$. A similar trend can be observed when the search window and neighborhood size increase, as shown in Table IV. The speedup goes from $222.43\times$, using a search window of 11×11 pixels and a neighborhood window of 11×11 pixels, to $1662.90\times$ using a search window of 41×41 pixels and a neighborhood window of 11×11 pixels. However, in this case the increment in the speedup is always greater as S and W increase. Two factors that may explain this phenomena are the improvement in the time complexity of DNLM-MA-P with respect to W and the better exploitation of the available processing resources of the KNL as the amount of work increases.

Table III
AVERAGE RUNTIME OF THE DNLM AND DNLM-MA-P FILTERS USING DIFFERENT IMAGE SIZES.

Image size	DNLM [s]	DNLM-MA-P [s]	Speedup [×]
256×256	10.26 ± 0.02	0.05 ± 0.00	206.95
512×512	40.95 ± 0.35	0.11 ± 0.00	377.70
1024×1024	163.18 ± 0.03	0.25 ± 0.01	662.27
2048×2048	654.15 ± 5.64	0.90 ± 0.01	728.66
4096×4096	2621.96 ± 23.00	3.45 ± 0.01	759.42
8192×8192	10539.96 ± 3.09	33.10 ± 0.61	318.38

Table IV
AVERAGE RUNTIME OF THE DNLM AND DNLM-MA-P FILTERS USING DIFFERENT SEARCH WINDOW SIZES s AND NEIGHBORHOOD WINDOW SIZES ω.

S	W	DNLM [s]	DNLM-MA-P [s]	Speedup [×]
11×11	5×5	42.46 ± 0.46	0.19 ± 0.00	222.43
	7×7	46.73 ± 0.41	0.20 ± 0.00	238.11
	11×11	61.61 ± 0.40	0.20 ± 0.00	306.38
21×21	5×5	147.96 ± 1.41	0.25 ± 0.00	595.86
	7×7	163.62 ± 1.41	0.25 ± 0.01	660.36
	11×11	218.32 ± 1.87	0.25 ± 0.00	880.76
41×41	5×5	485.15 ± 5.31	0.45 ± 0.01	1071.00
	7×7	546.90 ± 10.10	0.46 ± 0.01	1201.83
	11×11	752.35 ± 5.41	0.45 ± 0.00	1662.90

The effect of varying the number of threads on the parallel implementations of the DNLM filter is shown in Fig 2. The DNLM-MA-P implementation achieves its lowest execution time of 0.25 s using only 32 threads. When more threads are added the execution time is progressively higher, due to memory bound behavior of the DNLM-MA-P, as previously suggested in section III. However, independently of the number of threads, the DNLM-MA-P implementation is always faster than the DNLM-P and DNLM-IIFFT-P. On the other hand, the DNLM-IIFFT-P implementation achieves its lowest execution time of 1 s when using 128 threads. If we increase the number of threads to 256, the execution time increases to 1.63 s. Although the DNLM-P implementation is the slowest one for almost all the number of threads tested, it scales the best when the number of threads increases. Using 256 threads, the DNLM-P is faster than the DNLM-IIFFT-P and performs similar to the DNLM-MA-P implementation.

The efficiency of each parallel implementation of the DNLM filter is shown in Fig. 3. When the number of threads increases, the efficiency of the DNLM-MA-P filter rapidly decreases, being only 0.006 when using 256 threads. On the other hand, the efficiency of the DNLM-P and DNLM-IIFFT-P decreases more slowly. However, when using more than 64 threads, which is actually the number of physical cores of the KNL, the efficiency drops faster for both implementations. For the DNLM-IIFFT-P filter the efficiency goes from 0.86, when using 64 threads, to 0.18 when using 256 threads. For the DNLM-P it goes from 0.93, when using 64 threads, to 0.44 when using 256 threads.

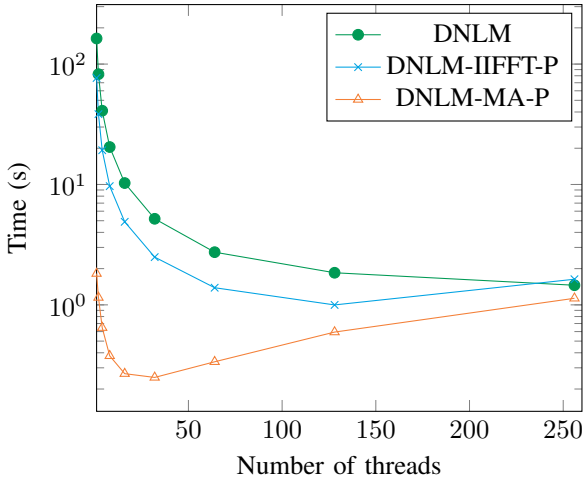


Figure 2. Effect of varying the number of threads on each parallel implementation of the DNLM filter.

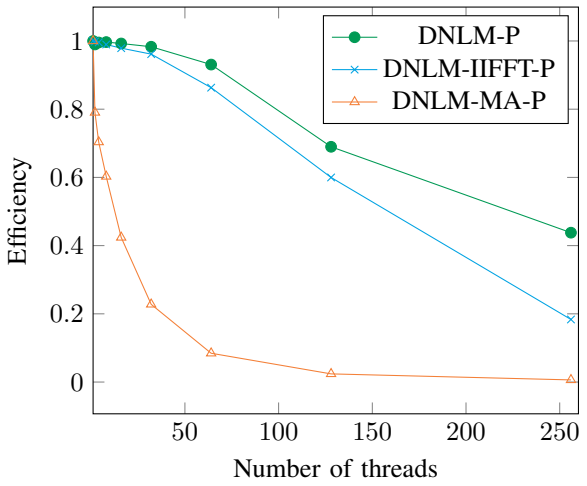


Figure 3. Efficiency of the parallel implementation for each DNLM filter version, from 1 to 256 threads.

V. CONCLUSIONS AND FUTURE WORK

The achieved speedup with the proposed DNLM-MA-P approach is stunning. It scales well with increasing number of available cores, and also the speedup is higher with larger data input and larger sizes of the search and neighborhood windows. The highest speedup achieved in this work (1662×), stands out over the previously reported speedups, summarized in Table I. The execution times of the DNLM-MA-P makes its usage practical for preprocessing large data sets, and improves latter image based pattern recognition tasks. For instance, preprocessing a data set of 170 000 images, a current need of microbiology researchers at the Universidad de Costa Rica to enhance latter cell segmentation and tracking, would take 11 hours. This result could be further enhanced by

using multi-node parallelism with MPI. Our approach overcomes the speedup reported in Table I for other Xeon Phi implementations, possibly due to effective vectorization and the improvement of KNL in contrast with previous Xeon Phi architecture.

Furthermore, the computational efficiency of the DNLM-MA-P allows its usage as a preprocessing stage in CNN architectures to improve the performance of different image based pattern recognition tasks. As previously addressed, recently in [6] authors found an impressive increase of a CNN model precision for age estimation using X-ray images using the DNLM as a preprocessing step. An efficient implementation of the DNLM makes its usage practical as a preprocessing stage in a CNN model, implementing parameter calibration to be performed during the CNN model training.

VI. Acknowledgements

This research was partially supported by a machine allocation on Kabré supercomputer at the Costa Rica National High Technology Center.

References

[1] P. Jain and V. Tyagi, "A survey of edge-preserving image denoising methods," *Information Systems Frontiers*, vol. 18, no. 1, pp. 159–170, 2016.

[2] S. Calderón, D. Moya, J. C. Cruz, and J. M. Valverde, "A first glance on the enhancement of digital cell activity videos from glioblastoma cells with nuclear staining," in *Central American and Panama Convention (CONCAPAN XXXVI), 2016 IEEE 36th*. IEEE, 2016, pp. 1–6.

[3] A. Sáenz, S. Calderón, J. Castro, R. Mora, and F. Siles, "Deceived bilateral filter for improving the automatic cell segmentation and tracking in the nf-kb pathway without nuclear staining," in *VI Latin American Congress on Biomedical Engineering CLAIB 2014, Paraná, Argentina 29, 30 & 31 October 2014*. Springer, 2015, pp. 345–348.

[4] R. Chacón-Quesada, S. Calderón-Ramírez, and F. Siles, "Improving the temporal segmentation in digital videos using the deceived bilateral filter," in *Central American and Panama Convention (CONCAPAN XXXVI), 2016 IEEE 36th*. IEEE, 2016, pp. 1–6.

[5] S. Dodge and L. Karam, "Understanding how image quality affects deep neural networks," in *Quality of Multimedia Experience (QoMEX), 2016 Eighth International Conference on*. IEEE, 2016, pp. 1–6.

[6] S. Calderon, F. Fallas, M. Zumbado, P. N. Tyrrell, H. Stark, Z. Emersic, B. Meden, and M. Solis, "Assessing the impact of the deceived non local means filter as a preprocessing stage in a convolutional neural network based approach for age estimation using digital hand x-ray images," in *International Conference on Image Processing 2018, in press*. IEEE.

[7] S. Calderón, A. Sáenz, R. Mora, F. Siles, I. Orozco, and M. Buemi, "Dewaff: A novel image abstraction approach to improve the performance of a cell tracking system," in *Bioinspired Intelligence (IWOBI), 2015 4th International Work Conference on*. IEEE, 2015, pp. 81–88.

[8] A. Buades, B. Coll, and J.-M. Morel, "Neighborhood filters and pdes," *Numerische Mathematik*, vol. 105, no. 1, pp. 1–34, 2006.

[9] S. C. Ramírez and M. Z. Corrales, "Dnlm-iifft: An implementation of the deceived non local means filter using integral images and the fast fourier transform for a reduced computational cost," in *Iberoamerican Congress on Pattern Recognition*. Springer, 2017, pp. 127–134.

[10] M. Mahmoudi and G. Sapiro, "Fast image and video denoising via nonlocal means of similar neighborhoods," *IEEE signal processing letters*, vol. 12, no. 12, pp. 839–842, 2005.

[11] A. Pardo, "Non Local Means Image Filtering Using Clustering," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, M. Mendoza and S. Velastín, Eds. Cham: Springer International Publishing, 2018, pp. 483–490.

[12] A. Dauwe, B. Goossens, H. Q. Luong, and W. Philips, "A fast non-local image denoising algorithm," in *Image Processing: Algorithms and Systems VI*, vol. 6812. International Society for Optics and Photonics, 2008, p. 681210.

[13] P. Chatterjee and P. Milanfar, "A generalization of non-local means via kernel regression," in *Computational Imaging VI*, vol. 6814. International Society for Optics and Photonics, 2008, p. 68140P.

[14] Z. Ji, Q. Chen, Q.-S. Sun, and D.-S. Xia, "A moment-based nonlocal-means algorithm for image denoising," *Information Processing Letters*, vol. 109, no. 23-24, pp. 1238–1244, 2009.

[15] T. Thaipanich, B. T. Oh, P.-H. Wu, and C.-C. J. Kuo, "Adaptive nonlocal means algorithm for image denoising," in *Consumer Electronics (ICCE), 2010 Digest of Technical Papers International Conference on*. IEEE, 2010, pp. 417–418.

[16] C. Karam and K. Hirakawa, "Monte-carlo acceleration of bilateral filter and non-local means," *IEEE Transactions on Image Processing*, vol. 27, no. 3, pp. 1462–1474, 2018.

[17] J. Wang, Y. Guo, Y. Ying, Y. Liu, and Q. Peng, "Fast non-local algorithm for image denoising," in *Image Processing, 2006 IEEE International Conference on*. IEEE, 2006, pp. 1429–1432.

[18] Y.-L. Liu, J. Wang, X. Chen, Y.-W. Guo, and Q.-S. Peng, "A robust and fast non-local means algorithm for image denoising," *Journal of Computer Science and Technology*, vol. 23, no. 2, pp. 270–279, 2008.

[19] B. Goossens, H. Luong, A. Pizurica, and W. Philips, "An improved non-local denoising algorithm," in *2008 International Workshop on Local and Non-Local Approximation in Image Processing (LNLA 2008)*, 2008, pp. 143–156.

[20] L. Condat, "A simple trick to speed up and improve the non-local means," 2010.

[21] J. Froment, "Parameter-free fast pixelwise non-local means denoising," *Image Processing On Line*, vol. 4, pp. 300–326, 2014.

[22] P. Coupé, P. Yger, and C. Barillot, "Fast non local means denoising for 3d mr images," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2006, pp. 33–40.

[23] J. Darbon, A. Cunha, T. F. Chan, S. Osher, and G. J. Jensen, "Fast nonlocal filtering applied to electron cryomicroscopy," in *Biomedical Imaging: From Nano to Macro, 2008. ISBI 2008. 5th IEEE International Symposium on*. IEEE, 2008, pp. 1331–1334.

[24] A. Márques and A. Pardo, "Implementation of non local means filter in gpus," in *Iberoamerican Congress on Pattern Recognition*. Springer, 2013, pp. 407–414.

[25] X. Mingliang, L. Pei, L. Mingyuan, F. Hao, Z. Hongling, Z. Bing, L. Yusong, and Z. Liwei, "Medical image denoising by parallel non-local means," *Neurocomputing*, vol. 195, pp. 117–122, 2016.

[26] B. Goossens, H. Luong, J. Aelterman, A. Pižurica, and W. Philips, "A gpu-accelerated real-time nlmeans algorithm for denoising color video sequences," in *International Conference on Advanced Concepts for Intelligent Vision Systems*. Springer, 2010, pp. 46–57.

[27] Y. Shi, X. Zhu, and R. Bamler, "Optimized parallelization of non-local means filter for image noise reduction of insar image," in *Information and Automation, 2015 IEEE International Conference on*. IEEE, 2015, pp. 1515–1518.

[28] T.-A. Nguyen, A. Nakib, and H.-N. Nguyen, "Medical image denoising via optimal implementation of non-local means on hybrid parallel architecture," *Computer methods and programs in biomedicine*, vol. 129, pp. 29–39, 2016.

[29] H. Zhu, Y. Wu, P. Li, D. Wang, W. Shi, P. Zhang, and L. Jiao, "A parallel non-local means denoising algorithm implementation with openmp and opencl on intel xeon phi coprocessor," *Journal of Computational Science*, vol. 17, pp. 591–598, 2016.

[30] F. Huang, B. Lan, J. Tao, Y. Chen, X. Tan, J. Feng, and Y. Ma, "A parallel nonlocal means algorithm for remote sensing image denoising on an intel xeon phi platform," *IEEE Access*, vol. 5, pp. 8559–8567, 2017.

[31] R. Gramunt, H.-s. Kim, and S. Hutsell, "Knights Landing:Second-Generation Intel Xeon Phi Product," *IEEE Micro*, vol. 36, no. 2, pp. 34–46, 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7453080/