Proceedings of the 2013 Federated Conference on
Computer Science and Information Systems pp. 495–498

3D Non-Local Means denoising via multi-GPU

Giuseppe Palma,
Marco Comerci
and Bruno Alfano
Inst. of Biostructures and Bioimaging
National Research Council of Italy
Via De Amicis 95, 80145 Naples, Italy
Email: giuseppe.palma@ibb.cnr.it

Salvatore Cuomo,
Pasquale De Michele
and Francesco Piccialli
Dept. of Mathematics and Applications
University of Naples Federico II
Via Cinthia, 80126 Naples, Italy
Email: salvatore.cuomo@unina.it

Pasquale Borrelli
Dept. of Advanced Biomedical Sciences
University of Naples Federico II
Via Pansini 5, 80131 Naples, Italy
Email: pasquale.borrelli@unina.it

***Abstract*—Non-Local Means (NLM) algorithm is widely considered as a state-of-the-art denoising filter in many research fields. High computational complexity led to implementations on Graphic Processor Unit (GPU) architectures, which achieve reasonable running times by filtering, slice-by-slice, 3D datasets with a 2D NLM approach. Here we present a fully 3D NLM implementation on a multi-GPU architecture and suggest its high scalability. The performance results we discuss encourage the coding of further filter improvements and the investigation of a large spectrum of applicative scenarios.**

I. Introduction

IMAGE denoising represents one of the most common tasks of image processing. Several techniques have been developed in the last decades to face the problem of removing noise from images, still preserving the small structures from an excessive blurring [2]. All those schemes share the belief that an improved value of a given image point can be expressed as a function of the image itself; each of them diverges in how the function is defined.

One of the most performing and robust denoising approaches is the non-local means (NLM) filter, introduced in [1]. Since its first appearance, the family of NLM algorithm and implementation variants has enormously grown (just to mention some of the most relevant improvements, see [5], [6], [7], [14], [10], [4], [11]); nevertheless, all of them assume that the restoring function for a given point is a mean of all the image values, largely weighted according to the radiometric similarity between values and only weakly tied to a spatial proximity criterion.

The result is a general-purpose denoising scheme, whose performances are widely accepted to be better with respect to the previous state-of-the-art algorithms, such as the total variation, the wavelet thresholding or the anisotropic filtering [13]. In particular, it has been shown that NLM filter guarantees the homogeneity of flat zones, preserves edges and fine structures, and transforms white noise into white noise, thus avoid the introduction of artifacts and spurious correlated signal [2].

Unsurprisingly, the NLM algorithm is computationally very heavy, and even some fast versions of the scheme are quite demanding on 2D images and almost daunting on 3D datasets. The huge amount of computational demand has been recently addressed by using accelerated hardware, the Graphic Processor Units (GPUs) in particular.

In 3D datasets, *e.g.* in the context of the Magnetic Resonance Imaging (MRI), the use of fully 3D filters is more appropriate than a 2D-based slice-by-slice filtering approach to exploit all the information contained in the image.

To the best of our knowledge, although there are several 2D GPU-based NLM versions ([9], [3], [8]), the 3D version of NLM filter has been poorly investigated in terms of both implementation and performance on GPUs.

In this paper, we present GPU and Multi-GPU versions of the 3D NLM filter based on Compute Unified Device Architecture (CUDA) [12]. We report the performance of the implementation for different 3D synthetic and real datasets. The parallelization of the filter via GPUs gives clinically-feasible MRI denoising execution times.

The plan of the paper is as follows. In §II we briefly describe the NLM algorithm. To follow, in §III we provide the implementation details. In §IV we present and discuss the results. Finally, in §V we draw conclusions and future works.

II. Theory

A. General description

An N-D image X can be considered as a real function $X : \mathbb{R}^N \to \mathbb{R}$ with a bounded support $\Omega \subset \mathbb{R}^N$. The NLM filter [1] is a class of endomorphisms of the image space, identified by 2 parameters (a and h), that acts as follows:

$$[\mathrm{NLM}_{a,h}(X)](\vec{x}) = Y(\vec{x}) = \frac{\int_\Omega \exp\left[-\frac{d_a^2(\vec{x},\vec{y})}{h^2}\right] X(\vec{y}) d\vec{y}}{\int_\Omega \exp\left[-\frac{d_a^2(\vec{x},\vec{y})}{h^2}\right] d\vec{y}}\,, \quad (1)$$

where

$$d_a^2(\vec{x},\vec{y}) \equiv \int_{\mathbb{R}^N} \left|X(\vec{x}+\vec{t}) - X(\vec{y}+\vec{t})\right|^2 \cdot \frac{\exp -\frac{\|\vec{t}\|^2}{2a^2}}{(2\pi)^{n/2}\cdot a} d\vec{t}. \quad (2)$$

The intensity of a given point of the new image is a mean of the intensities of the original image, according to a weight function that disregards any explicit criterion of spatial proximity and only considers a measure (ruled by h) of self-similarity between windows of radius a centered on each point (radiometric proximity).

If the image is defined on a discrete, regular grid $\left\{\vec{x}_i \,\middle|\, 1 \le i \le \prod_{l=1}^N L_l\right\}$,

$$X(\vec{x}) = \sum_i X_i \delta(\vec{x} - \vec{x}_i)\,, \quad (3)$$

978-1-4673-4471-5/$25.00

from Eqns. 1–2 it follows that the filtered dataset is given by

$$Y_i = \frac{\sum_j \exp\left[-\frac{d_a^2(\vec{x}_i,\vec{x}_j)}{h^2}\right] X_j}{\sum_j \exp\left[-\frac{d_a^2(\vec{x}_i,\vec{x}_j)}{h^2}\right]}, \tag{4}$$

$$d_a^2(\vec{x}_i,\vec{x}_j) = \sum_k \left| X(\vec{x}_i+\vec{\Delta}_k) - X(\vec{x}_j+\vec{\Delta}_k)\right|^2 \cdot \frac{\exp -\frac{\|\vec{\Delta}_k\|^2}{2a^2}}{(2\pi)^{n/2}\cdot a}. \tag{5}$$

Moreover, from Eqns. 4–5 it follows that the complexity of the filter is $O\left(\prod_{l=1}^{N} L_l^3\right)$.

B. Actual algorithm

Both computational issues and the convenience to introduce a geometric proximity criterion in addition to the pure radiometric distance measure led to a change in the original version of the NLM filter [7].

Therefore, given a search radius M, for each voxel i located at $\vec{x}_i$ we define a search box V_i as

$$V_i \equiv \left\{ \vec{x}_j \in \Omega \,|\, \|\vec{x}_j - \vec{x_i}\|_\infty < M \right\}. \tag{6}$$

The search box associated with the i-th voxel defines the ensemble of voxels whose intensities will be available in the following for restoring (denoising) of the intensity $X(\vec{x}_i)$, thus reducing the search freedom of Eqn. 4 (in that case, $V_i \equiv \Omega$). The authors of ([7], [10]) suggest that a good choice for M should guarantee the cardinality of the search box, $|V_i|$, to be of the order of 10^3.

Analogously, given a similarity radius d, for each voxel $\vec{x}_j$ within a given search box V_i, we can define a similarity box

$${}_jB_i \equiv \left\{ \vec{x}_k \in \Omega \,|\, \|\vec{x}_k - \vec{x_j}\|_\infty < d \right\}. \tag{7}$$

In this case, d plays the role of a in Eqn. 4, provided that the original smooth Gaussian kernel is replaced by a binary cut-off; a good choice for d should guarantee $|{}_jB_i| \sim 30$ ([7], [10]).

Finally, the denoised image is

$$Y_i = \frac{\sum_{\vec{x}_j \in V_i} \exp\left[-\frac{\|{}_jB_i - {}_iB_i\|_2^2}{h^2}\right] X_j}{\sum_{\vec{x}_j \in V_i} \exp\left[-\frac{\|{}_jB_i - {}_iB_i\|_2^2}{h^2}\right]}, \tag{8}$$

whence it results that the algorithm complexity is $O\left(|V_i|\cdot|{}_jB_i|\cdot\prod_{l=1}^{N} L_l\right)$.

The filter strength, which is determined by h, can be automatically tuned to obtain an optimized denoising, independently from the search radius M and the standard deviation of noise σ:

$$h^2 = 2\beta\sigma^2\,|V_i| \tag{9}$$

($\beta \sim 1$ is an adimensional constant to be manually tuned).

III. Implementation

General Purpose computation on Graphics Processing Units (GPGPU) is the use of GPUs to perform highly parallelizable computations that would normally be handled by CPU devices. Programming with GPUs requires both a deep understanding of the underlying computing architecture and a massive rethinking of existing CPU based algorithms.

A. Architecture

We implement the 3D NLM filter on the NVIDIA parallel computing architecture, which consists in a set of cores, or Scalar Processors (SPs), performing simple mathematical operations.

In the NVIDIA Fermi architecture, each SM has scheduler and dispatch units, execution units and a configurable memory of 64KB, which consists of a register file, an internal shared memory and an L1 cache. This memory is configurable in 16KB (or 48KB) for shared memory and 48KB (or 16KB) for L1 cache.

B. Mapping the algoritm on GPU

The Algorithm 1 is the pseudo-code of the NLM filter.

Algorithm 1 Pseudo-code of the NLM algorithm

```
1: for each voxel (i1, i2, i3) of the 3D image to be filtered do
2:    Initialize the cumulative sum of weights and the restored value to 0;
3:    for each voxel (j1, j2, j3) of the search window V_(i1,i2,i3) do
4:       for each voxel (k1, k2, k3) of the similarity window
         (j1,j2,j3)B_(i1,i2,i3) do
5:          Cumulate squared Euclidean distance;
6:       end for
7:       Calculate and cumulate the weight of the voxel in search window;
8:       Cumulate the restored value;
9:    end for
10:   Normalize restored value to the sum of the weights;
11: end for
```

In details, the statement at line 1 represents a nested iteration structure. In our GPU version, the loops on line 1 and line 3 in the Algorithm 1 are logically mapped onto the grid of thread blocks defined by means of the CUDA framework.

A first implementation in CUDA is presented in the Algorithm 2. Moreover, in order to make this algorithm compatible

Algorithm 2 CUDA code of NLM algorithm

```
1: int const i_1 = threadIdx.x +
     blockDim.x*blockIdx.x;
2: int const i_2 = threadIdx.y +
     blockDim.y*blockIdx.y;
3:   /* local statements */
4:   if ((i_1 ≥ 0) && (i_1 < X_Dim) &&      (i_2 ≥ 0)
   && (i_2 < Y_Dim)) {
5:      for(i_3=0; i_3 < Dim_Z; i_3++) {
6:        /* do something on img[i_1 +
            i_2*X_Dim + i_3*X_Dim*Y_Dim] */ }}
```

with multi-GPU architectures, we introduce some improvements. The number of GPU devices is returned by means of a CUDA library function and stored in the variable `n_gpus`. Then, the third dimension of the image is "splitted" between the available GPUs, setting the first (`start_k`) and the last (`end_k`) slices that each GPU has to manage. We report a sketch of the GPU implementation in the Algorithm 3.

In order to explore different types of data access, we test several configurations, both mono- and bi-dimensional, for the thread block size in which each slice is divided. Each thread processes sequentially the voxels along the third dimension. The workload is divided along the third dimension for multi-GPU configurations. Inside each GPU the workload

Algorithm 3 CUDA MULTI-GPU code of NLM algorithm

```
1: int const i_1 = threadIdx.x +
     blockDim.x*blockIdx.x;
2: int const i_2 = threadIdx.y +
     blockDim.y*blockIdx.y;
3:   /* split the image ``img'' between the
   ``n_gpus'' GPUs: each GPU works on the section
   of the image ``my_img'' */
4:   /* local statements */
5:   for(i_3 = 0; i_3 < Z_Dim/n_gpus;     i_3++) {
6:       /* do something on my_img[i_1 + i_2*X_Dim +
   i_3*X_Dim*Y_Dim/n_gpus] */
7:     } }
```

is divided along the first and second dimensions, in strips (mono-dimensional configurations) and tiles (bi-dimensional configurations) of threads. Strip or tile is allowed to cover entirely or only partially the slice grid.

We test also the impact of L1-cache on performance, using the binary L1-prefer setting, which allows to choose between two possible configurations: 48KB of shared memory and 16KB of L1-cache (no L1-prefer), or 16KB of shared memory and 48KB of L1-cache (L1-prefer).

The computing system is equipped with 2 Intel Xeon CPU E5620 (2.4 GHz) and an NVIDIA TESLA S2050 card. This device consists of 4 GPGPU units, each of which with 3GB of RAM memory and 448 cores at 1.15 GHz. The numerical code is implemented by using the single precision arithmetic. The CPU system is equipped with an Intel core i5-2500S (2.7-3.7 GHz).

IV. Results and Discussions

A. Consistency

As we aim to produce a strictly equivalent GPU implementation of the sequential NLM algorithm, we check the implementation consistency by comparing voxel-by-voxel the images obtained by one-core-CPU and GPU denoising. In Fig. 1 we show the 3D NLM filtering result on a real 3D knee MRI dataset. The difference between the GPU and CPU restored images falls within machine precision order of magnitude which are likely to be due to the arithmetic logic unit precision.

B. Performance

In order to investigate cache size impact on the execution time, we perform several test runs varying L1-prefer switch. Results are shown in Table I. L1-prefer choice gives a benefit on lager dataset, with a performance improvement ranging from fraction of percent in the smallest dataset to some 5% in the largest ones. These results suggest that the L1 miss rate, is low enough to have high performance processing even with old generation cards having small amount of cache.

The strip or tile thread division influences the performance of the filter in terms of computing time due to the different type of data access. Experimental results prove that optimal configuration is given by the strip subdivision. In Table I we report running times of (128,1,1) configuration on 2-GPU.

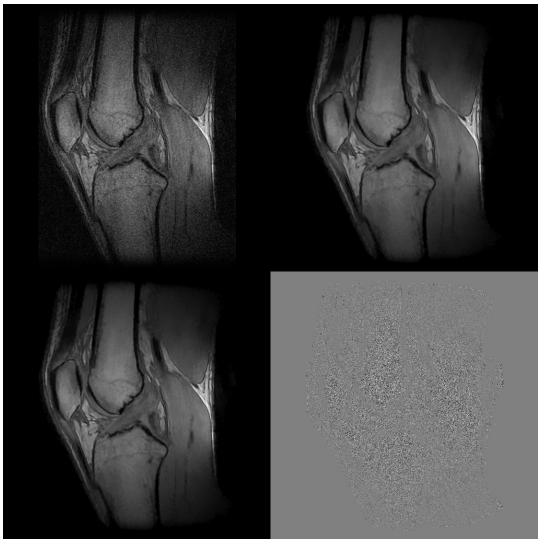

Fig. 1. From left to right and from top to bottom, the frames show a central slice of the original dataset, the GPU restored image, the CPU restored image and the difference between CPU and GPU filtered images (enhanced by a scaling factor of 10^6), respectively.

TABLE I
L1-prefer switch influence on execution times for (128,1,1) block size configuration and 3D random datasets.

	Cache configuration	
	L1-prefer	no L1-prefer
64x64x64	8.04	8.55
128x128x64	9.43	9.42
128x128x128	11.2	11.2
256x256x128	19.9	20.2
256x256x256	28.3	29.0
512x512x128	39.5	40.4
512x512x256	71.7	75.0
512x512x512	138	148

In Table II we report a comparison between running times of CPU, single GPU and multi-GPU implementation of 3D NLM filter with various thread block size. Reported running times include the overall data transfer between CPU and GPU and viceversa, which even for the biggest datasets appears negligible. Speed-up values suggest that the bigger the dataset to be filtered, the better the scalability of the implementation, which, for datasets size typical of MRI clinical practice, is close to be ideal. Moreover, the optimal thread size seems to be strips of thread between 128 and 256 elements. This result is consistent with NVIDIA guidelines [12]. Finally, on large datasets strip configuration should be prefered to tile configuration of the same size because more sequential memory access of the former. In Table III, we investigate the behavior of running times against the search ($|V_i|$) and similarity ($|_j B_i|$) window cardinalities. We note an high and almost constant speed-up among the various experiments, which makes feasible large window filter testing in a reasonable time.

Finally, in Figure 2 we outline the CPU, single GPU and multi-GPU GFlops for variable dataset sizes. It should be remarked that we are able to exploit up to 43.5% of single precision floating point peak performance of the GPU.

TABLE II
EXECUTION TIMES AND SPEED-UP VALUES FOR SEVERAL BLOCK SIZE CONFIGURATIONS AND 3D RANDOM DATASETS. SEARCH AND SIMILARITY WINDOWS HAVE BEEN SET ACCORDING TO $|V_i| = 11^3$ AND $|_jB_i| = 3^3$.

	Execution time/**Speed-up**								
	Single GPU				Multi-GPU				CPU
Dataset size	(16,16,1)	(128,1,1)	(256,1,1)	(512,1,1)	(16,16,1)	(128,1,1)	(256,1,1)	(512,1,1)	
64^3	5.08/**4.47**	5.73/**3.96**	5.63/**4.03**	6.94/**3.27**	11.7/**1.94**	8.04/**2.82**	8.53/**2.66**	12.6/**1.80**	22.7
128^2x64	6.48/**13.6**	7.30/**12.1**	7.08/**12.5**	10.2/**8.61**	8.97/**9.83**	9.43/**9.35**	9.31/**9.47**	10.8/**8.13**	88.2
128^3	9.06/**19.3**	10.8/**16.2**	10.4/**16.9**	16.7/**10.5**	10.3/**17.0**	11.2/**15.7**	10.9/**16.0**	14.0/**12.5**	175
256^2x128	22.1/**31.8**	24.3/**28.9**	25.0/**28.0**	31.0/**22.6**	16.9/**41.6**	19.9/**35.2**	18.6/**37.8**	21.1/**33.2**	702
256^3	40.5/**34.6**	44.7/**31.3**	47.2/**29.6**	59.4/**23.6**	26.0/**53.8**	28.3/**49.4**	29.5/**47.5**	35.1/**39.9**	1400
512^2x128	68.8/**41.0**	67.0/**42.1**	67.1/**42.0**	72.7/**38.8**	40.5/**69.6**	39.5/**71.4**	40.0/**70.5**	42.4/**66.5**	2820
512^2x256	136/**41.2**	132/**42.6**	131/**42.8**	142/**39.5**	73.8/**76.3**	71.7/**78.5**	72.0/**78.2**	77.5/**72.6**	5630
512^3	277/**40.7**	268/**42.2**	264/**42.7**	285/**39.6**	142/**79.3**	138/**82.0**	137/**82.4**	148/**76.2**	11300

TABLE III
EXECUTION TIMES AND SPEED-UP VALUES FOR A 3D RANDOM DATASET (SIZE = $512 \times 512 \times 128$) FOR SEVERAL WINDOW CONFIGURATIONS.

	Execution time / **Speed-up**								
	Single GPU				Multi-GPU				CPU
$(\|V_i\|, \|_jB_i\|)$	(16,16,1)	(128,1,1)	(256,1,1)	(512,1,1)	(16,16,1)	(128,1,1)	(256,1,1)	(512,1,1)	
$(11^3, 3^3)$	68.8/**41.0**	67.0/**42.1**	67.1/**42.0**	72.7/**38.8**	40.5/**69.6**	39.5/**71.4**	40.0/**70.5**	42.4/**66.5**	2820
$(21^3, 3^3)$	447/**44.4**	434/**45.7**	434/**45.7**	467/**42.4**	228/**87.0**	222/**89.4**	221/**89.6**	239/**82.8**	19800
$(11^3, 5^3)$	235/**34.6**	221/**36.7**	223/**36.6**	255/**31.9**	123/**61.1**	116/**69.9**	117/**69.3**	130/**62.4**	8140
$(21^3, 5^3)$	1650/**35.5**	1510/**38.8**	1520/**38.7**	1820/**32.2**	817/**72.0**	757/**77.6**	764/**77.0**	906/**64.8**	58800

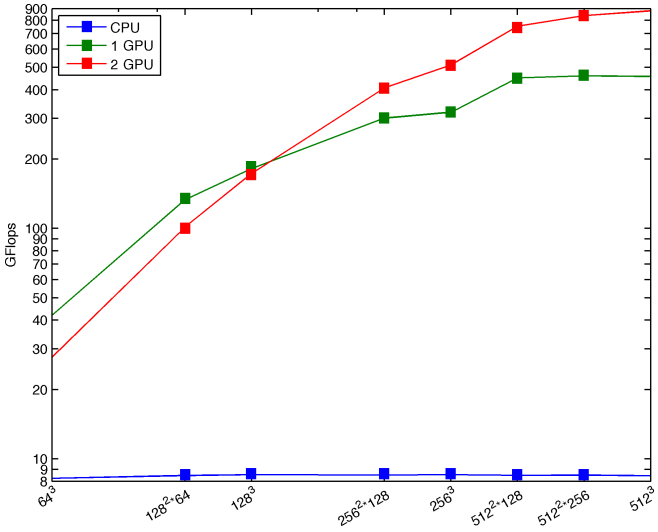

Fig. 2. Outline of the CPU, single GPU and multi GPU GFlops values for different dataset sizes. Please note the logarithmic scale of the axes.

V. CONCLUSIONS

NLM filter is a state-of-the-art denoising algorithm. However, the huge amount of computational load prevents the large-scale diffusion of its most common implementations.

To the best of our knowledge, in this paper we presented the first multi-GPU implementation of a fully 3D NLM filter. We analyzed several configurations of thread block organization and data access, thus identifying a set of optimal settings that guarantee high performance results for a wide spectrum of application scenarios. The reduction of running times shows that scalability is close to ideal one for most common dataset sizes, *e.g.* those typical of MRI clinical pratice. Speed-up high values encourage the exploration of more sophisticated algorithm variants, and reduce the gap between the previous execution times and acceptable performance for real-time scenarios.

REFERENCES

[1] A. Buades, B. Coll, J.M. Morel, *A review of image denoising algorithms, with a new one*, Multiscale Model. Simul. 4, 490-530 (2005).
[2] A. Buades, B. Coll, J.M. Morel, *Image Denoising Methods. A New Nonlocal Principle*, SIAM Review. 52, 113-147 (2010).
[3] F.P.X. De Fontes, G.A. Barroso, P. Coupé, P. Hellier, *Real time ultrasound image denoising*, J. Real-Time Image Process 6, 15-22 (2011).
[4] K. Dabov, A. Foi, V. Katkovnik K. Egiazarian, *Image Denoising by Sparse 3-D Transform-Domain Collaborative Filtering*, Image Processing, IEEE Transactions on, 2080-2095 (2007).
[5] M. Ebrahimi, E. Vrscay, *Examining the role of scale in the context of the non-local-means filter*, in Image Analysis and Recognition, LNCS. vol. 5112, pp. 170-181, Springer-Verlag, Berlin (2008).
[6] H. Xu, J. Xu, F. Wu, *On the biased estimation of nonlocal means filter*, Proceedings of IEEE IC on Multimedia and Expo, pp. 1149-1152 (2008).
[7] P. Coupé, P. Yger, S. Prima, P. Hellier, C. Kervrann, C. Barillot, *An Optimized Blockwise Nonlocal Means Denoising Filter for 3-D Magnetic Resonance Images*, IEEE Trans. Med. Imag. 27, 425-441 (2008).
[8] B. Goossens, Q. Luong, J. Aelterman, A. Pizurica and W. Philips, *A GPU-accelerated real-time NLMeans algorithm for denoising color video sequences*, Proceedings of 12th IC on Advanced Concepts for Intelligent Vision System pp. 46-57, 6475 of LNCS, Springer (2010).
[9] K. Huang, D. Zhang, K. Wang *Non-local means denoising algorithm accelerated by GPU*, SPIE Conference Series (2009).
[10] J.V. Manjón, N.A. Thacker, J.J. Lull, G. Garcia-Martí, L. Martí-Bonmatí, M. Robles, *Multicomponent MR Image Denoising*, IJBI, 2009.
[11] M. Maggioni, V. Katkovnik, K. Egiazarian, A. Foi, *Nonlocal Transform-Domain Filter for Volumetric Data Denoising and Reconstruction*, IEEE Transactions on Image Processing, 119-113 (2013).
[12] *NVIDIA CUDA programming guide* http://developer.download.nvidia.com, Nvidia Technical Report, 2012.
[13] H. Seo, P. Chatterjee, H. Takeda, P. Milanfar, *A comparison of some state of the art image denoising methods*, in Proceedings of the 41st Asilomar Conference on Signals, Systems, and Computers (2007).
[14] N. Wiest-Daesslé, S. Prima, P. Coupé, S. Morrissey, C. Barillot, *Rician noise removal by non-local means filtering for low signal-to-noise ratio MRI: Applications to DT-MRI*, MICCAI, LNCS. vol. 5242, pp. 171-179, Springer-Verlag, Berlin (2008).