Int J Parallel Prog (2018) 46:528–542
https://doi.org/10.1007/s10766-017-0505-2

A GPU Implementation of OLPCA Method in Hybrid Environment

Pasquale De Michele[1] · **Francesco Maiorano**[1] · **Livia Marcellino**[2] · **Francesco Piccialli**[1]

Received: 28 February 2017 / Accepted: 23 April 2017 / Published online: 3 May 2017

Abstract Sophisticated denoising algorithms are used to improve image quality in the Magnetic Resonance Imaging field. Of course, better results are obtained by implementing computationally expensive schemes. In this paper, we consider the Overcomplete Local Principal Component Analysis (OLPCA) method for image denoising and its main issues. More in detail, we investigated the impact of the Singular Value Decomposition on the OLPCA algorithm and its high computational cost. Moreover, we propose a fine-to-coarse parallelization strategy in order to exploit a parallel hybrid architecture and we implement a multilevel parallel software as a smart combination between codes using NVIDIA cuBLAS library for Graphic Processor Units (GPUs) and the standard Message Passing Interface library for cluster programming. Experimental results show improvements in terms of execution time with a promising speed up with respect to the CPU and our old GPU versions.

Keywords Overcomplete local principal component analysis · High performance computing · Graphic processor units · Hybrid architectures

✉ Livia Marcellino
livia.marcellino@uniparthenope.it

Pasquale De Michele
pasquale.demichele@unina.it

Francesco Maiorano
frances.maiorano@studenti.unina.it

Francesco Piccialli
francesco.piccialli@unina.it

1 University of Naples Federico II, Strada Vicinale Cupa Cintia 21, 80126 Naples, Italy

2 University of Naples Parthenope, Centro Direzionale, Isola C4, 80143 Naples, Italy

1 Introduction

A common task in the Magnetic Resonance Imaging (MRI) field is image denoising. Over the last few years, several techniques have been proposed to face the image noise removal problem, for small structures without excessive blurring (see [4,7], for details). Among these methods, some performing and robust denoising approaches are the Joint Linear Minimum Mean Squared Error (JLMMSE) [30], Non-local Means (NLM) [3,15,26], Adaptive Non-local Means (ANLM) [24], Principal Component Analysis (PCA) [5,28] and Local PCA (LPCA) [25]. For completeness, other approaches are based on the minimization of a suitable cost function [6,18,20,21] . In this context, Diffusion Weighted Imaging (DWI) has received much attention because of its feature to measure the microscopic motion of water molecules within tissue.

In this paper, we choose the LPCA scheme, in order to exploit the multi-directional redundancy of the multicomponent Diffusion Weighted (DW) images. In particular, we resort to the Overcomplete LPCA (OLPCA) method, discussed in [23], where for each pixel in an image, local estimates are combined in order overlap little portions of the image. However, the huge amount of computational resources demanded by this method is a critical issue.

We started from previous works [9,10], where we have designed and implemented a hybrid CPU and Graphic Processor Unit (GPU) parallel version of the OLPCA algorithm. Indeed, recently on the track of current developments of GPUs for High Performance Computing (HPC), many algorithms has been accelerated, with good results [8,14,17,19,22,27].

To be specific, we have chosen to parallelize some basic modules by using Compute Unified Device Architecture (CUDA)[1] on a GPU architecture, combining the *shared* and *global* memories. The GPUs are massively parallel architectures that efficiently work with impressive performance improvements. However, the use of GPUs requires a deep understanding of the underlying architecture and ad hoc thread based algorithms.

Unfortunately, in our previous software there was a bottleneck for the overall parallel application: the most expensive linear algebra task in OLPCA algorithm, the Singular Value Decomposition (SVD), was processed on the CPU. Clearly, we observed that making the SVD computation in a sequential manner on CPU, caused a slowdown which inevitably affects performance. Generally, the SVD is a very difficult routine to parallelize. To have an idea of possible parallelization strategies, we refer to [2].

Furthermore, in another work [12], our aim has been to perform an SVD profiling in a GPU-CUDA environment in order to analyse how to speed up the SVD computational task. We investigated how cuBLAS and CULA optimized libraries could support our analysis. We reported tests for a group of SVDs showing execution times of a SVD performed on a CPU (Intel Core i7 860 @ 2.80 Ghz) and a GPU (NVIDIA Quadro

[1] http://www.nvidia.com/.

K5000 4 Gb). In the GPU version, we performed a SVD decomposition comparing two libraries: CULA[2] and cuBLAS.[3]

GPU performance gains are noticeable when the squared input matrix size increases, while the CPU version performs better when the problem size is small. However, in the OLPCA algorithm, the SVD is performed on many small matrices. Once again, this critical issue seems to encourage a sequential implementation for the OLPCA method as many of them can be performed simultaneously with a proper approach.

In addiction, we resume an idea proposed in [11,13] and the recent trend to adopt hybrid architectures, i.e. High Performance Computing (HPC) environments [1], where special purpose devices are chosen with the aim to improve performance of the overall application [16,29]. More in details, we resort to a hybrid architecture based on a cluster of multi-processors environment, equipped with GPU devices. Therefore, we propose to exploit the multilevel parallel paradigm for such hybrid computing system, in order to implement a novel parallel algorithm for the OLPCA method.

In our paradigm, the parallel implementation resorts to: (*i*) the cuBLAS library, to perform basic linear algebra operations and the SVD computation; (*ii*) the Message Passing Interface (MPI) library[4] for the domain decomposition, to overtake the space limitation issue of GPUs. In this way, we design a fine-to-coarse parallelization by suggesting a coarse-grained strategy, which enforces concurrency among cluster nodes, by means of the problem decomposition. Conversely, to compute local problems, we use a fine-grained parallelization strategy using suitable cuBLAS routines.

Finally, we prove that the proposed multilevel parallel framework is able to leverage the computational power of the hybrid architecture adopted, on its full capability. More precisely, the proposed strategy allows us to calculate the SVD necessary for the DWI denoising algorithm in a much shorter time as opposed to the sequential approach and we report some tests to prove it.

The paper is organized as follows. In Sect. 2 we give a short description of the OLPCA algorithm and our previous parallel implementations. Then, in Sect. 3, we provide implementation details for our novel hybrid parallel software. Moreover, results of our work are reported and discussed in Sect. 4. Finally, in Sect. 5 we draw conclusions.

2 The Parallel OLPCA Method

The OLPCA method is used to perform the denoising of sequences of DW images (usually affected by Rician noise). They have a multi-directional structure generating 4D sequences along axes (x, y, z and k), where the last dimension displays the k directions of the microscopic motion for each slice (see Fig. 1).

The OLPCA method takes advantage of the multi-directional redundancy of that multicomponent and works as follows: let x_p be a voxel of a three-dimensional DW image, the 3D patches, surrounding x_p in any direction k, are reordered as a column

[2] http://www.culatools.com/.

[3] https://developer.nvidia.com/cublas.

[4] http://www.mcs.anl.gov/research/projects/mpi/.

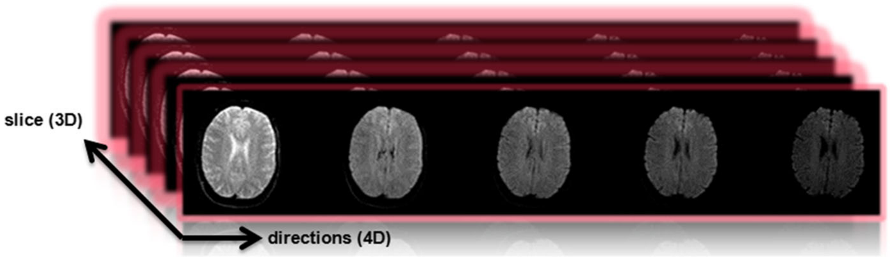

Fig. 1 DWI sequence

Algorithm 1 The LPCA scheme

```
1: for each $X$ do
2:     $C = X^T X$                          //covariance matrix,
3:     $C = WDW^T$                          //singular value decomposition
4:     $Y = XW$
5:     $\hat{Y} = threshold(Y, \tau)$       //thresholding step
6:     $\hat{X} = \hat{Y} W^T$              //denoised output
7: end for
```

vector of a matrix X with $N \times K$ components, where N are the patch items and K the directions. The scheme of the LPCA method is listed in the Algorithm 1.
More in detail:

– the matrix W contains the eigenvectors, while the diagonal matrix D contains the eigenvalues
– a SVD is computed for each matrix X of size $N \times K$,
– the threshold τ is equal to the local noise variance level $\gamma \cdot \sigma$
– the matrices W^T and W^{-1} are equal because W is an orthogonal matrix.

Finally, the Overcomplete rule, for each voxel, combines all the estimates as follows:

$$\hat{x_i} = \sum_j \frac{1}{1 + ||\hat{Y_j}||_0}. \qquad (1)$$

In a previous work [10], we have proposed a parallel implementation of the OLPCA method, which exploits (i) the natural multidimensional structure of the DW images and (ii) the numerous floating point operations of the OLPCA algorithm itself.

Here, we summarize, as much as possible, only the basic concepts about our parallel OLPCA implementation. It consists of some GPU-parallel modules of the algorithm on top of a three-dimensional computational grid, as big as the 3D basis of the DW image. This way, a voxel (i, j, z, k) is associated to each thread, in all its k directions.

In Fig. 2 we can observe the three-dimensional computational grid of threads, coloured in green, while the $4D$ DW image is represented by cyan cubes (each cube is a direction of the image). A single thread in the grid is coloured in red, while its competence voxels, in the DW image, are highlighted in orange in the k available directions.

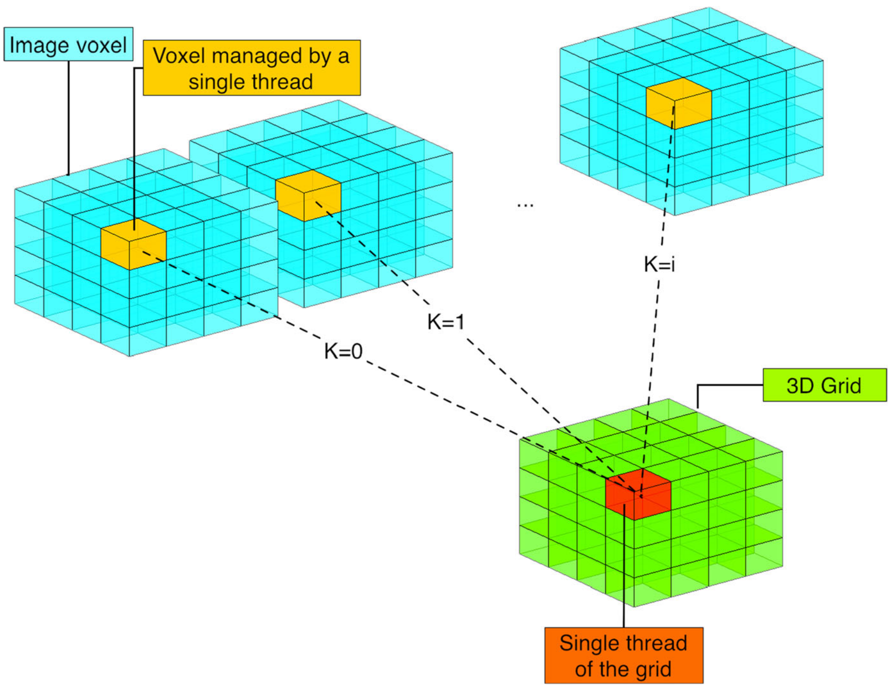

Fig. 2 Distribution thread with competence elements

This parallel algorithm combines use of the *shared* and *global* memories in the OLPCA computational tasks. In this way, although the access times for the global memory are higher than the shared memory, we can benefit the absence of space constraints. More in detail, the algorithm uses four CUDA kernels for removing Rician noise in DW images.

(i) The first kernel constructs the matrix X locally and computes both the mean of each column of X and the covariance matrix for each voxel $(i,\ j,\ z)$. Data are all declared and computed in the shared memory. Afterwards they are moved to the global memory.
(ii) The second kernel, using a combination of shared and global memories, computes the product between the matrices X and W (in the shared memory) and assigns the result to the matrix Y (defined in the global memory), on which a threshold is then applied.
(iii) The third kernel computes the inverse PCA, reconstructing, in the global memory, the restored matrix $\hat{X}$ by using eigenvectors moved from shared to global memory. Afterwards the kernel performs the overlapping.
(iv) Finally, the last kernel computes, entirely in global memory, the weighted mean for each voxel in the restored output image.

Notice that for all kernels the block size is:

$$nThreadPerBlock.x \times nThreadPerBlock.y \times nThreadPerBlock.z,$$

while the grid dimension is:

$$\left\lfloor \frac{\text{LENGTH}}{nThreadPerBlock.x} \right\rfloor \times \left\lfloor \frac{\text{WIDTH}}{nThreadPerBlock.y} \right\rfloor \times \left\lfloor \frac{\text{DEPTH}}{nThreadPerBlock.z} \right\rfloor,$$

where LENGTH and WIDTH are the number of rows and columns, respectively, and DEPTH is the depth of the DW image. Moreover, the structures used are tailored with a dimension able to contain the data for each thread in all the k directions.

The SVD task in this software was processed on the CPU, causing a slowdown which inevitably affects the performances. In other words, this GPU approach is affected by a bottleneck found in the OLPCA SVD software and a few considerations have to be addressed to solve this issue. In the next section we will illustrate how we bypassed this problem.

3 Hybrid Implementation

Here we propose a solution to the bottleneck problem of our previous parallel algorithm, exploiting the hierarchical parallelism of novel architectures such as a cluster of multi-processors equipped with GPUs. More in detail, we use a GPUs' cluster, which is still a challenge, for a lack in terms of efficient and general programming model approaches. Observe that, GPUs' clusters can provide high peak performance at small cost, and so their importance and spreading are currently increasing among the scientific software community.

Figure 3 shows a hybrid architecture consisting of n CPUs and GPUs. The CPUs (in the green boxes) and the GPUs (in the orange boxes), each of which having its own storage, communicate via an I/O Hub. More in detail, CPUs consist of more cores, each of them with an L1 cache memory, and any couple of cores share an L2 cache memory.

Furthermore, a basic organization of GPUs is comprised of a set of cores, or Scalar Processors (SPs), performing simple mathematical operations and organized into a streaming multiprocessor (SM). Each SM has a shared memory, a shared L1-cache, and several other units. We highlight that such an architecture has many advantages. Firstly, when the GPU performs a computation, the CPU simultaneously executes computing operations, rather than waiting the GPU. Secondly, it is possible to observe a reduction of the number of communicating nodes (i.e., the total communications). This feature depends on the number of processors per number of cores in itself. Here, the most difficult challenge is to avoid the communications among the cluster nodes due to exchanges of data and, simultaneously, solving local sub-problems as efficiently as possible.

Our paradigm is based on a fine-to-coarse strategy in order to implement an efficient multilevel parallel framework. The main feature of our idea is to follow a domain decomposition approach, while, inside each node, the local computational kernels can be solved by using specific libraries as middleware. To be specific, the multilevel paradigm is composed by two nested parallelization strategies:

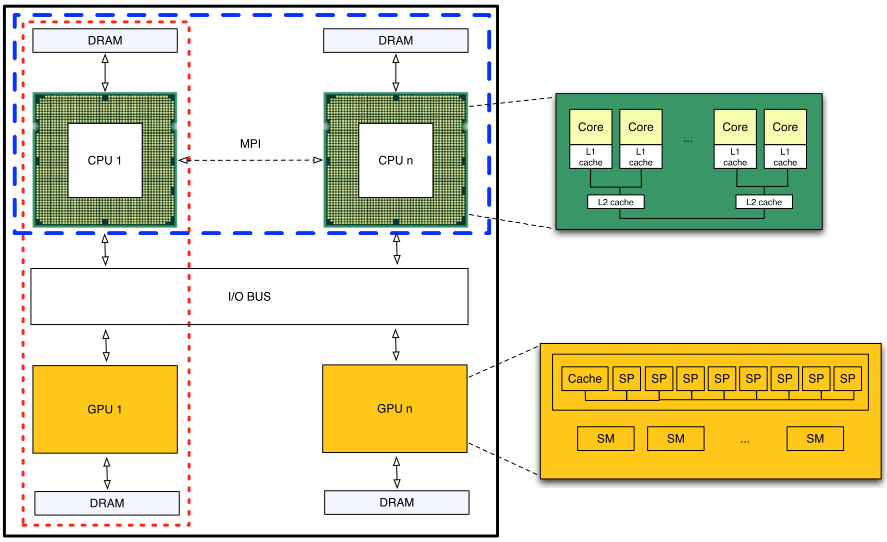

Fig. 3 Hybrid architectures: a cluster of multiprocessors composed of *n* many-core CPUs and GPUs

– The *intra-node* configuration, which implements a coarse-grained parallelization strategy on multiprocessor systems. The computing elements involved in this strategy are reported in Fig. 3, in the blue broken box. This approach relies on a suitable domain decomposition with a coarse-grained parallelism, which exploits the intra-node concurrency of multi-processors in order to distribute among nodes the matrices X, constructed for each voxel, as described in the previous section. Notice that the data distribution and collection is carried out by a direct *scattering-gathering*, with MPI functions.
– The *inter-node* configuration realizes a fine-grained parallelization strategy on a single node with a GPU. The computing elements involved in this strategy are reported in Fig. 3, in the red dotted box. Here, we compute the local OLPCA algorithms exploiting the massive parallelism of the GPU inside a single node by means of the CUDA-Nvidia environment.

Finally, combining these two configurations, we have our fine-to-coarse parallelization strategy on a GPUs' clusters, which allows us to implement our multilevel framework and to achieve significant benefits. This strategy minimizes the communication among the nodes (data must be only distributed at the beginning and collected at the end of local operations), and especially plans the local sub-problems solution by means of a specific middleware (for some well known computational kernel, appropriate optimized libraries can be used).

3.1 Multi-level Approach

In order to illustrate the implementation details, we start from the description of the OLPCA algorithm done in Sect. 2.

The matrices X, for each voxel x_p, are built on one cluster node and on GPU environment, as illustrated at point (i) in the previous section. Therefore, their processing can be distributed to nodes in the cluster. To be specific, we have a matrix X, for each voxel $x_p,\ p = 1,\ \ldots,\ L \times W \times D$, where:

$$L = \mathit{LENGHT},\ W = \mathit{WIDTH},\ D = \mathit{DEPTH},$$

are the number of rows, columns and the depth of the DW image, respectively.

Suppose we have a cluster of μ processors, then the $L \times W \times D$ matrices will be scattered to all processors. This means that each node (with ID: Id) will be working with $nloc$ matrices, where:

$$nloc = \begin{cases} \left\lceil \frac{L\times W\times D}{\mu} \right\rceil & \mathit{if}\quad Id < mod(L \times W \times D, \mu) \\ \\ \left\lfloor \frac{L\times W\times D}{\mu} \right\rfloor & \mathit{if}\quad Id \geq mod(L \times W \times D, \mu) \end{cases}$$

More in details, we use the `MPI_Scatter` and `MPI_Broadcast` basic routines of the MPI library, to distribute data among processors, following the *inter-node* parallel approach.

Then, each node should execute the Algorithm 1 on GPU environment, for $nloc$ matrices. Following the *intra-node* parallel approach, each node computes the tasks (ii), (iii) and (iv) on our GPU, as described in Sect. 2. Moreover, here we add a new GPU-parallel task for the SVD computation of a high number of C matrices of small size, which will be described in the next subsection.

Finally, results are collected by means of the `MPI_Gather` routine of the MPI library, following the *inter-node* parallel approach and using the Overcomplete rule defined in (1).

3.2 SVD Factorization by Batches CuBLAS Routine

Here, we focus on how to parallelize a specific class of problems, where we request a solution for a group of identical problems, namely a Single Instruction Multiple Data (SIMD) operation. Such operations leverage data-level parallelism and allow to focus on distributing data across different nodes in order to achieve multi-level parallelism. Moreover, the SVDs are independent to each other. Organizing this problem in batches is the logic next step.

In such a case study, a way of exposing parallelism on a GPU is to perform the same cuBLAS routine on multiple independent problems simultaneously. While it is possible to achieve this by executing multiple cuBLAS kernels over several CUDA streams, batched cuBLAS routines naturally allow this type of parallelism for specific operations (GEMM, GETRF, GETRI, TRSM and GEQRF). In the past few years, NVIDIA introduced such batched operations in its cuBLAS and cuSOLVER libraries.

As we pointed out in [12], these interfaces are best suited for particular circumstances, that is large batch sizes (number of matrices > 1000) and small matrix size

$$A = \overset{\text{Initial Matrix}}{\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix}} \Longrightarrow \overset{\text{Bidiagonal Form}}{\begin{bmatrix} x & x & 0 & 0 & 0 \\ 0 & x & x & 0 & 0 \\ 0 & 0 & x & x & 0 \\ 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & x \end{bmatrix}} \Longrightarrow \overset{\text{Diagonal Form}}{\begin{bmatrix} x & 0 & 0 & 0 & 0 \\ 0 & x & 0 & 0 & 0 \\ 0 & 0 & x & 0 & 0 \\ 0 & 0 & 0 & x & 0 \\ 0 & 0 & 0 & 0 & x \end{bmatrix}}$$

Fig. 4 Golub and Reinsch factorization scheme

(matrix size < 32). Our case study relates to this category of problems. Our batch size consists of more than 650.000 matrices each of size 14×14. Unfortunately, due to lacking of a SVD batched routine in the cuBLAS and cuSOLVER libraries, we decided to leverage the aforementioned batched routines, specifically *cublas<t>geqrfBatched* for QR factorization, and implement the Golub and Reinsch (1970) algorithm to ultimately obtain a SVD of multiple small matrices.

Briefly described in Fig. 4 is the Golub and Reinsch algorithm to obain a SVD of a matrix A. It consists of two main steps:

– Convert the input matrix to a bidiagonal form using Householder transformations.
– Diagonalize the resulting matrix through QR transformations.

The key computational task in this case is the QR factorization. cuBLAS batched QR routine has 8 parameters: (i) a handle to cuBLAS, (ii) and (iii) the width and height of the matrices (14×14), (iv) an array `Aarray` of pointers to matrices stored in a column-major format, (v) the `lda` dimension, (vi) an array `TauArray` of pointers to vectors of dimension $max(1, min(m, n))$, (vii) an output parameter which is zero if input parameters are valid, less than zero otherwise and (viii) the batch size, namely the number of pointers in `Aarray`. For each `Aarray[i]`, for $i = 0, \ldots,$ `batchSize`-1, the *cublas<t>geqrfBatched* routine performs a QR factorization using Householder reflections. Each output matrix `Q[i]` is stored in the lower part of each `Aarray[i]` and is represented by the product of elementary reflectors. As stated by cuBLAS documentation, this function is intended to be used for matrices of small sizes where the launch overhead is a significant factor. cuBLAS batched operations accept matrices of arbitrary dimension but only support a compute capability of 2.0 or above.

4 Experiments

The computing environment we used to develop, execute and test the MultiLevel GPU-parallel OLPCA algorithm is a cluster in which each node is equipped with a CPU Intel Core i7 (2.8 GHz), 8 GB of RAM memory and a GPU Nvidia Quadro K5000, with 1536 CUDA cores, 2.1 TFLOPS in single precision, 4 GB GDDR5 of memory size and 173 GB/s of memory bandwidth, Network: 1 Gb Ethernet, Switch CISCO Layer3.

In the following we describe the tests performed and the results achieved in terms of accuracy and performance. Quality of denoising is assessed by using as reference

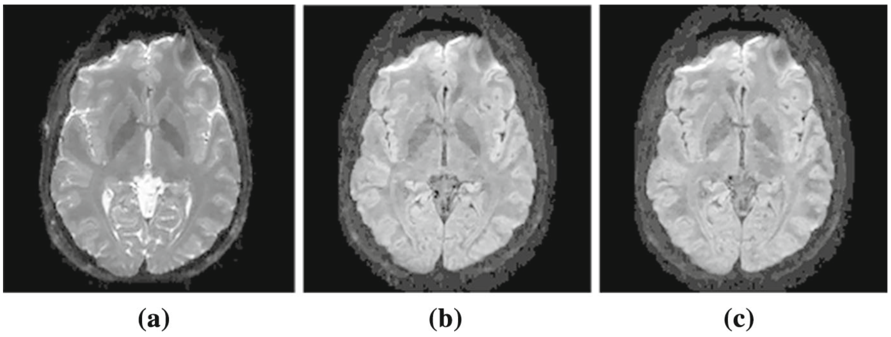

Fig. 5 The ground-truth images used

Table 1 Noisy images with depth $z = 0$ and direction $k = 0$

Noise level (%)	Threshold factor	Truth-noise	Truth-denoise
Image $k = 0$			
3	3.5	39.3627	40.4992
5	4	35.1357	36.0129
7	5.5	32.5834	33.3899
9	5.5	30.6760	32.8739

a sample image free of noise, namely a ground-truth,[5] in NIFTI format. Starting from this image, various percentages of *Rician noise* were added, in order to obtain noisy images to be restored. DW images used in this work have size $176 \times 176 \times 21 \times 10$, where: 10 are the directions (i.e. 6504960 voxel) and the LPCA method is able to denoise all the images for each z and k, where z is the depth of the diffusion weighted image and k is the direction. In the accuracy tests, we reported results related to a fixed $z = 0$ in the directions $k = 0,\ 5,\ 9$. In Fig. 5, the ground-truth images we have chosen are shown.
The metric used for the quality is Peak Signal-Noise Ratio (PSNR):

$$\text{PSNR} = 10 \log_{10} \frac{L^2}{MSE},$$

where L is the maximum voxel value and MSE is the mean square error. PSNR values for the two sample images used, obtained varying the noise, are reported in the following.
By observing Tables 1, 2 and 3, in which each row represents an image with a fixed noise level (3, 5, 7 and 9%), it is possible to note that when noise level increases, the PSNR between ground-truth and noise tends to decrease. Moreover, in conditions of equal noise, the PSNR values in the last columns of Tables 2 and 3 are higher than those reported in Table 1. This is due to the fact that the DW images have a signal/noise

[5] Available at http://brainweb.bic.mni.mcgill.ca/brainweb/.

Table 2 Noisy images with depth $z = 0$ and direction $k = 5$

Noise level (%)	Threshold factor	Truth-noise	Truth-denoise
Image $k = 5$			
3	3.5	40.0504	46.4654
5	4	36.5721	43.7716
7	5.5	33.606	42.1731
9	5.5	31.471	40.4865

Table 3 Noisy images with depth $z = 0$ and direction $k = 9$

Noise level (%)	Threshold factor	Truth-noise	Truth-denoise
Image $k = 9$			
3	3.5	40.2151	46.8808
5	4	36.6531	44.0942
7	5.5	33.7361	42.1145
9	5.5	31.5565	40.5904

Table 4 Execution times (in seconds) for the SVD task, depth $z = 0$ and direction $k = 5$

SVD (CPU): 650496	QR (GPU): 5003	SVD (GPU): 5003
Image $k = 5$		
35.17	1.23	14.25

ratio lower than other kinds of images. Ultimately, the PSNR shows that the algorithm produces output results with a good accuracy from a numerical point of view. In Fig. 6 we reported the case $z = 0, k = 0,\ 5,\ 9$ and noise level equal to 3%.

In the following we show the results about the performance tests. We first report tests performed on the core computational task, namely the SVD (see Table 4). We remind that for one direction k, the dataset size is: $176 \times 176 \times 21$, i.e. 650496 voxels. For each of them a SVD factorization must be computed. Here we report the execution time only for direction $k = 5$, but the execution times for other directions k is it's always the same, because the computation remain unchanged.

The CPU version is based on the SVD implementation provided by the OpenCV library; while, in the GPU version, cuBLAS QR batched routine represents the core of this particular batch computation. More in details, we considered 130 (μ) nodes in our cluster so to achieve a batch size of approximately 5000 matrices for each node. We then compared CPU and GPU performance for such batch size, taking into account that to have an SVD the Golub and Reinsch method needs to perform approximately 10 QR decomposition.

We derived the following: to process 650, 000 SVD in CPU environment requires more time than 5000×130 SVD on GPU environment with our algorithm, because these are performed simultaneously. In other words, while GPU performance is worse than the CPU version on such a batch size, exploiting a hybrid architecture and spread

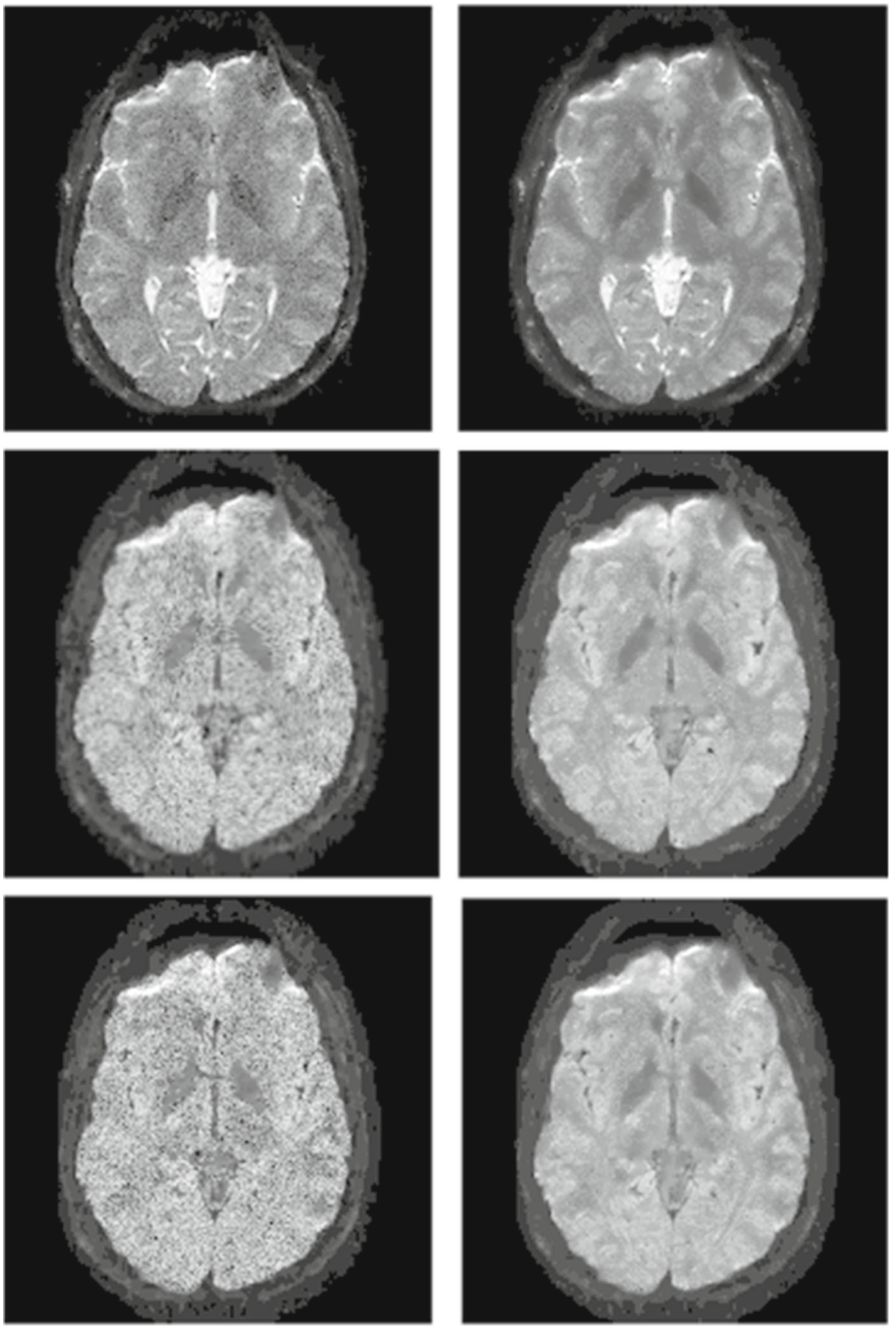

Fig. 6 *Top* noisy (3%) and denoised images with $z = 0$ and $k = 0$. *Middle* noisy (3%) and denoised images with $z = 0$ and $k = 5$. *Bottom* noisy (3%) and denoised images with $z = 0$ and $k = 9$

Table 5 Execution times (in seconds) for OLPCA, depth $z = 0$ and direction $k = 5$

CPU	CPU+GPU	ML_GPU
Image $k = 5$		
52.17	36.54	19.62

650, 000 matrices on this hybrid solution allows us to achieve better performance than the sequential CPU version: decompositions are computed in the same amount of time one node computes a batch of 5000. Moreover, with proper use of device memory, increasing the batch size more parallelism could be exploited.

Finally, in Table 5, we report the comparison between our old CPU−GPU parallel software (CPU+GPU), the serial version on CPU and the novel hybrid multi-level parallel software (ML_GPU).

5 Conclusions

In order to provide a fast denoising algorithm for DW images, we presented a hybrid GPU-parallel software based on the OLPCA method. Our approach uses a fine-to-coarse parallelization strategy in order to exploit a parallel hybrid architecture in which codes using NVIDIA cuBLAS library for GPUs and the standard MPI library for cluster programming are combined. The proposed multilevel parallel framework is able to leverage the computational power of the hybrid architecture adopted, minimizing communication among nodes and solving the local sub-problems by a specific optimized library. Our results show significant improvements in terms of performances with respect to the CPU version. As a future development, we will investigate on the implementation of a multiple SVD GPU kernel where each thread executes an independent decomposition of a matrix. In this a way data locality could be better exploited, and probably overhead could be minimized, improving also the speed-up.

References

1. Abate, D., Ambrosino, F., Aprea, G., Bastianelli, T., Beone, F., Bertini, R., Bracco, G., Calosso, B., Caporicci, M., Chinnici, M., Colavincenzo, A., Cucurullo, A., D'Angelo, P., De Michele, P., De Rosa, M., Del Giudice, E., Funel, A., Furini, G., Giammattei, D., Giusepponi, S., Guadagni, R., Guarnieri, G., Italiano, A., Magagnino, S., Mariano, A., Mencuccini, G., Mercuri, C., Migliori, S., Ornelli, P., Palombi, F., Pecoraro, S., Perozziello, A., Pierattini, S., Podda, S., Poggi, F., Ponti, G., Quintiliani, A., Rocchi, A., Scio, C., Simoni, F., Vita, A.: The role of medium size facilities in the hpc ecosystem: the case of the new cresco4 cluster integrated in the eneagrid infrastructure. In: International Conference on High Performance Computing and Simulation, pp. 1030–1033, HPCS 2014, Bologna, Italy, 21–25 July (2014). doi:10.1109/HPCSim.2014.6903807
2. Berry, M., Sameh, A.: Special issue on parallel algorithms for numerical linear algebra an overview of parallel algorithms for the singular value and symmetric eigenvalue problems. J. Comput. Appl. Math. **27**(1), 191–213 (1989). doi:10.1016/0377-0427(89)90366-X
3. Buades, A., Coll, B., Morel, J.: A review of image denoising algorithms, with a new one. Multiscale Model. Simul. **4**(2), 490–530 (2005). doi:10.1137/040616024
4. Buades, A., Coll, B., Morel, J.: Image denoising methods. A new nonlocal principle. SIAM Rev. **52**(1), 113–147 (2010). doi:10.1137/090773908

5. Bydder, M., Du, J.: Noise reduction in multiple-echo data sets using singular value decomposition. Magn. Reson. Imaging **24**(7), 849–856 (2006). doi:10.1016/j.mri.2006.03.006. http://www.sciencedirect.com/science/article/pii/S0730725X06001317
6. Cafieri, S., D'Apuzzo, M., De Simone, V., Di Serafino, D., Toraldo, G.: Convergence analysis of an inexact potential reduction method for convex quadratic programming. J. Optim. Theory Appl. **135**(3), 355–366 (2007). doi:10.1007/s10957-007-9264-3
7. Campagna, R., Crisci, S., Cuomo, S., De Michele, P., Galletti, A., Marcellino, L., Murano, A.: A novel split Bregman algorithm for MRI denoising task in an e-health system. In: ACM International Conference Proceeding Series. Proceedings of the 9th PETRA Conference will held on the Island of Corfu, Greece at the Corfu Holiday Palace Hotel from June 29 to July 1 (2016). doi:10.1145/2910674.2910692. http://dl.acm.org/citation.cfm?doid=2910674.2910692
8. Cuomo, S., De Michele, P., Galletti, A., Marcellino, L.: A gpu-parallel algorithm for ecg signal denoting based on the nlm method. In: 30th IEEE International Conference on Advanced Information Networking and Applications, AINA 2016, Crans-Montana, Switzerland, March 23–25, 2016, pp. 35–39 (2016). doi:10.1109/WAINA.2016.110. http://doi.ieeecomputersociety.org/10.1109/WAINA.2016.110
9. Cuomo, S., De Michele, P., Galletti, A., Marcellino, L.: A gpu parallel implementation of the local principal component analysis overcomplete method for dw image denoising. In: 2016 IEEE Symposium on Computers and Communication (ISCC), pp. 26–31 (2016). The Twenty-First IEEE Symposium on Computers and Communication, 27–30 June 2016, Messina, Italy. doi:10.1109/ISCC.2016.7543709
10. Cuomo, S., De Michele, P., Galletti, A., Marcellino, L.: Local principal component analysis overcomplete method: a gpu parallel implementation combining shared and global memories. In: International Conference on High Performance Computing and Simulation, HPCS 2016, Innsbruck, Austria, July 18–22, 2016, pp. 81–87 (2016). doi:10.1109/HPCSim.2016.7568319
11. Cuomo, S., De Michele, P., Galletti, A., Marcellino, L.: A parallel pde-based numerical algorithm for computing the optical flow in hybrid systems. J. Comput. Sci. (2017). doi:10.1016/j.jocs.2017.03.011
12. Cuomo, S., De Michele, P., Maiorano, F., Marcellino, L.: Advances on P2P, parallel, grid, cloud and internet computing. Lecture Notes on Data Engineering and Communications Technologies, vol. 1, chap. GPU Profiling of Singular Value Decomposition in OLPCA Method for Image Denoising, pp. 707–716. Springer International Publishing (2017). doi:10.1007/978-3-319-49109-7_68. Proceedings of the 11th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing 3PGCIC-2016 November 5–7, 2016, Soonchunhyang University, Asan, Korea. Online ISBN: 978-3-319-49109-7
13. Cuomo, S., Galletti, A., Giunta, G., Marcellino, L.: Toward a multi-level parallel framework on gpu cluster with petsc-cuda for pde-based optical flow computation. pp. 170–179 (2015). doi:10.1016/j.procs.2015.05.220. http://www.scopus.com/inward/record.url?eid=2-s2.0-84939155665&partnerID=40&md5=ddcb2162cbc29925e582fc9498463059
14. Cuomo, S., Galletti, A., Marcellino, L.: A gpu algorithm in a distributed computing system for 3d MRI denoising. In: F. Xhafa, L. Barolli, F. Messina, M. R Ogilla (eds.) 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, Krakow, Poland, pp. 557–562, November 4–6 (2015). doi:10.1109/3PGCIC.2015.77
15. Cuomo, S., Michele, P.D., Piccialli, F.: 3d data denoising via nonlocal means filter by using parallel GPU strategies. Comput. Math. Methods Med. **523**, 1–523. doi:10.1155/2014/523862
16. D'Amore, L., Arcucci, R., Marcellino, L., Murli, A.: A parallel three-dimensional variational data assimilation scheme. AIP Conf. Proc. **1389**(1), 1829–1831 (2011). doi:10.1063/1.3636965
17. D'Amore, L., Laccetti, G., Romano, D., Scotti, G., Murli, A.: Towards a parallel component in a gpucuda environment: a case study with the l-bfgs harwell routine. Int. J. Comput. Math. **92**(1), 59–76 (2015). doi:10.1080/00207160.2014.899589
18. de Angelis, P.L., Bomze, I.M., Toraldo, G.: Ellipsoidal approach to box-constrained quadratic problems. J. Glob. Optim. **28**(1), 1–15 (2004). doi:10.1023/B:JOGO.0000006654.34226.fe
19. D'Amore, L., Marcellino, L., Mele, V., Romano, D.: Deconvolution of 3d fluorescence microscopy images using graphics processing units. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **7203 LNCS**(PART 1), pp. 690–699 (2012). doi:10.1007/978-3-642-31464-3_70
20. De Asmundis, R., di Serafino, D., Hager, W., Toraldo, G., Zhang, H.: An efficient gradient method using the yuan steplength. Comput. Optim. Appl. **59**(3), 541–563 (2014). doi:10.1007/s10589-014-9669-5

21. Gmez, S., Severino, G., Randazzo, L., Toraldo, G., Otero, J.: Identification of the hydraulic conductivity using a global optimization method. Agric. Water Manag. **96**(3), 504–510 (2009). doi:10.1016/j.agwat.2008.09.025
22. Laccetti, G., Lapegna, M., Mele, V., Romano, D.: A study on adaptive algorithms for numerical quadrature on heterogeneous gpu and multicore based systems. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **8384 LNCS**(PART 1), pp. 704–713 (2014). doi:10.1007/978-3-642-55224-3_66
23. Manjón, J., Coupé, P., Concha, L., Buades, A., Collins, D., Robles, M.: Diffusion weighted image denoising using overcomplete local pca. PLoS ONE **8**(9) (2013). doi:10.1371/journal.pone.0073021. http://www.scopus.com/inward/record.url?eid=2-s2.0-84883366803&partnerID=40&md5=467a3af41b50d17486ab1385ccf8e816
24. Manjón, J.V., Coupé, P., Martí-Bonmatí, L., Collins, D.L., Robles, M.: Adaptive non-local means denoising of MR images with spatially varying noise levels. J. Magn. Reson. Imaging **31**(1), 192–203 (2010). doi:10.1002/jmri.22003. http://www.hal.inserm.fr/inserm-00454564
25. Muresan, D.D., Parks, T.W.: Orthogonal, exactly periodic subspace decomposition. IEEE Trans. Signal Process. **51**(9), 2270–2279 (2003). doi:10.1109/TSP.2003.815381
26. Palma, G., Piccialli, F., Michele, P.D., Cuomo, S., Comerci, M., Borrelli, P., Alfano, B.: 3d non-local means denoising via multi-gpu. In: Proceedings of the 2013 Federated Conference on Computer Science and Information Systems, Kraków, Poland, September 8–11, 2013, pp. 495–498 (2013). http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6644045
27. Piccialli, F., Cuomo, S., De Michele, P.: A regularized mri image reconstruction based on hessian penalty term on cpu/gpu systems, pp. 2643–2646 (2013). doi:10.1016/j.procs.2013.06.001. http://www.scopus.com/inward/record.url?eid=2-s2.0-84892506892&partnerID=40&md5=cc785a43da0426b134b5a4e05bc3ad5e
28. Poon, P., Wei-Ren, N., Sridharan, V.: Image denoising with singular value decompositon and principal component analysis. http://www.u.arizona.edu/~ppoon/ImageDenoisingWithSVD.pdf (2009)
29. Song, F., Dongarra, J.: A scalable approach to solving dense linear algebra problems on hybrid cpu–gpu systems. Concurr. Comput. **27**(14), 3702–3723 (2015). doi:10.1002/cpe.3403
30. Tristán-Vega, A., Aja-Fernández, S.: DWI filtering using joint information for DTI and HARDI. Med. Image Anal. **14**(2), 205–218 (2010). doi:10.1016/j.media.2009.11.001