Parallel Hyperspectral Coded Aperture for Compressive Sensing on GPUs

Sergio Bernabé, Gabriel Martín, José M. P. Nascimento, José M. Bioucas-Dias, *Member, IEEE*, Antonio Plaza, *Senior Member, IEEE*, and Vítor Silva

***Abstract*—The application of compressive sensing (CS) to hyperspectral images is an active area of research over the past few years, both in terms of the hardware and the signal processing algorithms. However, CS algorithms can be computationally very expensive due to the extremely large volumes of data collected by imaging spectrometers, a fact that compromises their use in applications under real-time constraints. This paper proposes four efficient implementations of hyperspectral coded aperture (HYCA) for CS, two of them termed P-HYCA and P-HYCA-FAST and two additional implementations for its constrained version (CHYCA), termed P-CHYCA and P-CHYCA-FAST on commodity graphics processing units (GPUs). HYCA algorithm exploits the high correlation existing among the spectral bands of the hyperspectral data sets and the generally low number of endmembers needed to explain the data, which largely reduces the number of measurements necessary to correctly reconstruct the original data. The proposed P-HYCA and P-CHYCA implementations have been developed using the compute unified device architecture (CUDA) and the cuFFT library. Moreover, this library has been replaced by a fast iterative method in the P-HYCA-FAST and P-CHYCA-FAST implementations that leads to very significant speedup factors in order to achieve real-time requirements. The proposed algorithms are evaluated not only in terms of reconstruction error for different compressions ratios but also in terms of computational performance using two different GPU architectures by NVIDIA: 1) GeForce GTX 590; and 2) GeForce GTX TITAN. Experiments are conducted using both simulated and real data revealing considerable acceleration factors and obtaining good results in the task of compressing remotely sensed hyperspectral data sets.**

***Index Terms*—Coded aperture, compressive sensing (CS), graphics processing units (GPUs), high-performance computing, hyperspectral imaging.**

Manuscript received January 21, 2015; revised April 17, 2015; accepted May 11, 2015. Date of publication June 10, 2015; date of current version February 09, 2016. This work was supported by the Portuguese Science and Technology Foundation under Projects UID/EEA/50008/2013 and SFRH/BPD/94160/2013.

S. Bernabé, G. Martín, and J. M. Bioucas-Dias are with the Instituto de Telecomunicações, Instituto Superior Técnico, 1049-001 Lisbon, Portugal.

J. M. P. Nascimento is with the Instituto Superior de Engenharia de Lisboa, Instituto Politécnico de Lisboa, Lisbon 1959-007, Portugal and also with Instituto de Telecomunicações, Lisbon, Portugal.

A. Plaza is with the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura, E-10003 Cáceres, Spain.

V. Silva is with the Instituto de Telecomunicações, Departamento de Engenharia Electrotécnica e de Computadores, Universidade de Coimbra, 3030-290 Coimbra, Portugal.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSTARS.2015.2436440

I. Introduction

HYPERSPECTRAL imaging instruments allow data collection in hundreds or even thousands of spectral bands (at different wavelength channels) for the same area on the surface of the Earth [1]. For instance, NASA is continuously gathering imagery data with instruments such as the Jet Propulsion Laboratory's Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS), which is able to record the visible and near-infrared spectrum (wavelength region from 0.4 to 2.5 μm) of reflected light in an area of 2 to 12 km wide and several kilometers long, using 224 spectral bands [2]. The resulting multidimensional data cube typically comprises several GBs per flight. As a result, the computational requirements needed to store, manage, and process these images are enormous [3].

An important problem in the analysis of hyperspectral data is the presence of mixed pixels [4], which arise when the low spatial resolution of the sensor is not enough to separate spectrally distinct materials. Mixed pixels can also result when distinct materials are combined into a homogeneous or intimate mixture [5]. The spectra of the individual materials which forms the mixed pixel are often called endmembers in the hyperspectral imaging literature [6]. A challenging task in hyperspectral imagery, called hyperspectral unmixing [4], aims at determining the endmember signatures present in the data and estimating their abundance fractions within each pixel. In recent years, several approaches have been proposed to solve the aforementioned problem using high-performance computing systems such as commodity clusters [7], which is a solution very difficult to adapt to on-board systems. Other solutions based on low-weight integrated components, such as field programmable gate arrays (FPGAs) [8], [9] are easier to adapt to on-board systems. However, since algorithms are often in progress and in course of optimization, together with the long design and programmability times of FPGA technology, suggest its employment only when the mathematical treatment will be well consolidated. Possible alternatives are multicore processors [10] and commodity graphics processing units (GPUs) [11], which offer highly relevant computational power at low cost, thus offering the opportunity to bridge the gap toward real-time analysis of remotely sensed hyperspectral data [12]–[15].

Owing to the extremely large volumes of data collected by imaging spectrometers, hyperspectral data compression has received considerable interest in recent years [16], [17], mainly due to their capability to simplify the hardware and software requirements of the hyperspectral acquisition systems [18]–[21]. These data are usually acquired by a satellite or

an airborne instrument and sent to a ground station on Earth for subsequent processing. Usually, the bandwidth connection between the satellite/airborne platform and the ground station is reduced, which limits the amount of data that can be transmitted. As a result, there is a clear need for (either lossless or lossy) hyperspectral data compression techniques that can be applied onboard the imaging instrument [22]–[24]. In contrast, compressive sensing (CS) [25], [26] involves acquisition of the data in an already compressed form by computing inner products, also termed measurements, between known spectral vectors and the original data. This process is sometimes called "coded aperture" because inner products can be conceived as the total amount of light that is transmitted trough masks acting on the aperture of the instrument. In CS, the original data is inferred from the measurements by solving a convex optimization problem. A necessary condition to obtain good inferences is that the original data admits a sparse or compressible representation in a given basis or frame. This means that most of the coefficients of the representation in that basis or frame are zero or small and, thus, the data can be well approximated with just a small number of large coefficients. It happens that hyperspectral images are often highly compressible owing to a very high spatial and spectral correlation. Therefore, this imaging modality is a perfect candidate to apply the CS technology. Until now, there has been no effort to accelerate coded aperture algorithms for hyperspectral images using parallel techniques in the open literature.

In this paper, four computationally efficient implementations of an HYCA algorithm for CS on GPU platforms are proposed. HYCA [27] algorithm and its constrained version (CHYCA) are two algorithms that have been shown to be very successful from the viewpoint of using CS for improving the acquisition process of hyperspectral scenes. In this work, these algorithms are performed with several optimizations for accelerating their computational performance while maintaining their accuracy. The first one exploits the GPU architecture at low level, using shared memory and coalesced accesses to memory. The second one is an optimization focused on the use of a fast iterative method to solve a quadratic problem, avoiding the use of cuFFT library. The obtained new fast versions are called P-HYCA-FAST and P-CHYCA-FAST. The third one is focused on the configuration of a larger level one (L1) cache size and a smaller shared memory in the kernel developed on the P-HYCA-FAST and P-CHYCA-FAST implementations that lead to very significant speedup factors, thus taking full advantage of the computational power of GPUs. The considered implementations are intercompared in the context of real hyperspectral imaging applications. NVidia GeForce GTX 590 and GTX TITAN platforms have been used to test the proposed implementations with both synthetic and real hyperspectral scenes. Our study reveals that the NVidia GeForce GTX TITAN GPU can provide real-time CS performance. The implementations on GPUs have been carried out using NVidia CUDA and the cuBLAS library.[1] The cuFFT library[2] is only used on P-HYCA and P-CHYCA versions.

[1][Online]. Available: http://developer.nvidia.com/cuBLAS.
[2][Online]. Available: http://developer.nvidia.com/cuFFT.

This paper is organized as follows. Section II describes the original HYCA, CHYCA, and the proposed fast optimization for both methods. Section III describes the proposed GPU implementations. Section IV presents an experimental evaluation of the proposed implementations in terms of both accuracy and parallel performance using synthetic and real hyperspectral data sets on two GPU platforms. Finally, Section V concludes with some remarks and hints at plausible future research lines.

II. Description of the Methods

In this section, HYCA and CHYCA methods are introduced and described in the following sections. Moreover, a fast optimization for both cases is proposed to accelerate the CS process.

A. HYCA Algorithm

The original HYCA method for CS was developed in [27]. This approach compresses the data on the acquisition process, then the compressed signal is sent to Earth and stored in compressed form. Later, the original signal can be recovered by taking advantage of two key properties of hyperspectral images: 1) the spectral vectors live systematically in low-dimensional subspaces [28]; and 2) the spectral bands present a high correlation in the spatial domain. The former property allows to represent the data vectors using a reduced set of spectral endmembers due to the mixing phenomenon and also exploits the high spatial correlation of the fractional abundances associated to the spectral endmembers.

Let $\mathbf{x}_i \in \mathbb{R}^{n_b}$, for $i = 1, \ldots, n_p$, denote the $n_p := n_r \times n_c$ spectral vectors of a hyperspectral image, where n_r, n_c, and n_b denote, respectively, the number of rows, columns, and bands of the hyperspectral image, and $\mathbf{x} := [\mathbf{x}_1, \ldots, \mathbf{x}_{n_p}]^T$ [the operator $()^T$ stands for transpose] denote, in a vector format, the hyperspectral image. In order to perform the compression of the original signal $\mathbf{x}$, and as in [27], for each pixel $i \in \{1, \ldots, n_p\}$, a set of q inner products between $\mathbf{x}_i$ and samples of i.i.d. Gaussian random vectors is performed. The total number of measurements is therefore $q \times n_p$ yielding an undersampling factor of q/n_b. This measurement operation can be represented as a matrix multiplication

$$\mathbf{y} = \mathbf{A}\mathbf{x} \tag{1}$$

where $\mathbf{A}$ is a block-diagonal matrix containing the matrices $\mathbf{A}_i \in \mathbb{R}^{q \times n_b}$ acting on the pixel $\mathbf{x}_i$, for $i \in \{1, \ldots, n_p\}$. For reasons linked with 1) the computational management of the sampling process and 2) the spatial correlation length of hyperspectral images (see [27] for more details), matrices $\mathbf{A}_i$ are organized into spatial windows of size $ws \times ws = m$. Each window contains the same set of matrices. All windows have the same spatial configuration of $\mathbf{H}_j$, for $j = 1, \ldots, m$.

The HYCA method also takes advantage of the fact that the hyperspectral vectors $\mathbf{x}_i$ generally live in a low-dimensional subspace. This fact can be modeled by $\mathbf{x}_i = \mathbf{E}\mathbf{z}_i$, where $\mathbf{E} \in \mathbb{R}^{n_b \times p}$ is a matrix whose columns spans the signal subspace and $\mathbf{z}_i \in \mathbb{R}^p$ denotes the vector of coordinates with respect to the

columns of $\mathbf{E}$. Defining $\mathbf{z} := [\mathbf{z}_1^T, \ldots, \mathbf{z}_{n_p}^T]^T$, for $i = 1, \ldots, n_p$, we have

$$\mathbf{x} = (\mathbf{I} \otimes \mathbf{E})\mathbf{z} \tag{2}$$

where $\mathbf{E} \in \mathbb{R}^{n_b \times p}$, with $p \ll n_b$, $\otimes$ stands for Kronecker product, and $\mathbf{I}$ is the identity matrix of suitable size. In this work, we assume that the linear mixing model is a good approximation to the spectral vectors $\mathbf{x}_i$ [4] and, therefore, matrix $\mathbf{E}$ contains in its columns the spectral signatures of the p endmembers. We use the VCA algorithm [29] to infer $\mathbf{E}$. Since $\mathbf{E}$ is the mixing matrix, hence $\mathbf{z}$ contains the fractional abundances associated to each pixel.

Let $\mathbf{K} = \mathbf{A}(\mathbf{I} \otimes \mathbf{E})$. If matrices $\mathbf{E}$ and $\mathbf{A}$ are available, one can formulate the estimation of $\mathbf{z}$ from $(q \times n_x)$-dimensional vector of measurements. Since the fractional abundances in hyperspectral images exhibit a high spatial correlation, we exploit this feature for estimating $\mathbf{z}$ using the following optimization problem:

$$\begin{aligned} &\min_{\mathbf{z}} (1/2)\|\mathbf{y} - \mathbf{K}\mathbf{z}\|^2 + \lambda_{TV}\mathrm{TV}(\mathbf{z}) \\ &\text{subject to :} \quad \mathbf{z} \geq \mathbf{0} \end{aligned} \tag{3}$$

where $\mathrm{TV}(\mathbf{z})$ stands for the sum of nonisotropic total variations (TVs) [30], [31] associated to $\mathbf{z}$, one per image of abundance. Defined as

$$\mathrm{TV}(\mathbf{z}) := \boldsymbol{\phi}(\mathbf{D}\mathbf{z})$$

where $\mathbf{D} := [\mathbf{D}_h^T \mathbf{D}_v^T]^T$, $\mathbf{D}_h, \mathbf{D}_v$ compute the horizontal and vertical backward differences, assuming a cyclic boundary, and

$$\boldsymbol{\phi}(\boldsymbol{\vartheta}) := \sum_{i=1}^{p} \sum_{j=1}^{n_p} \|\boldsymbol{\vartheta}[i, j]\|$$

with $\boldsymbol{\vartheta} := [\boldsymbol{\vartheta}_h^T, \boldsymbol{\vartheta}_v^T]$, $\boldsymbol{\vartheta}_h$ standing for horizontal differences and $\boldsymbol{\vartheta}_v$ standing for vertical differences. The TV regularizer promotes piecewise abundance images $\mathbf{z}$. Therefore, the minimization (2) aims at finding a solution which is a compromise between the fidelity to the measured data, enforced by the quadratic term $(1/2)\|\mathbf{y} - \mathbf{K}\mathbf{z}\|^2$, and the properties enforced by the TV regularizer, that is piecewise smooth image of abundances. The relative weight between the two characteristics of the solution is set the regularization parameter $\lambda_{TV} > 0$.

To solve the convex optimization problem in (3), a methodology closely related with the one presented in [32] is adopted. The solution of this problem is obtained by an instance of the alternating direction method of multipliers (ADMM) [33], which decomposes very hard problems into a cyclic sequence of simpler problems. With this in mind, an equivalent way of writing the optimization problem in (3) is

$$\min_{\mathbf{z}} \frac{1}{2}\|\mathbf{y} - \mathbf{K}\mathbf{z}\|^2 + \lambda_{TV}\ \boldsymbol{\phi}(\mathbf{D}\mathbf{z}) + \iota_{R+}(z) \tag{4}$$

where $\iota_{R+}(\mathbf{z}) = \sum_{i=1}^{pn_p} \iota_{R+}(\mathbf{z}_i)$ is the indicator function ($\mathbf{z}_i$ represents the ith element of $\mathbf{z}$ and $\iota_{R+}(\mathbf{z}_i)$ is zero if $\mathbf{z}_i$ belongs to the nonnegative orthant and $+\infty$ otherwise). Given the objective function in (4), we can write the following equivalent formulation:

$$\begin{aligned} \min_{\mathbf{z},\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3,\mathbf{v}_4} & \frac{1}{2}\|\mathbf{y} - \mathbf{K}\mathbf{v}_1\|^2 + \iota_{R+}(\mathbf{v}_2) + \lambda_{TV}\ \boldsymbol{\phi}(\mathbf{D}\mathbf{z}) \\ \text{subject to} \quad & \mathbf{v}_1 = \mathbf{z} \\ & \mathbf{v}_2 = \mathbf{z} \\ & (\mathbf{v}_3, \mathbf{v}_4) = \mathbf{D}\mathbf{z}. \end{aligned} \tag{5}$$

Algorithm 1 shows the pseudocode of the HYCA algorithm to solve the problem in (5) and how to reconstruct the data using (2).

Algorithm 1. Pseudocode of HYCA algorithm

1. **Initialization**: set $k = 0$, choose $\mu > 0$, $\mathbf{E}$, $\mathbf{z}^{(\mathbf{0})}$, $\mathbf{v}_1^{(0)}, \mathbf{v}_2^{(0)}, \mathbf{v}_3^{(0)}, \mathbf{v}_4^{(0)}, \mathbf{d}_1^{(0)}, \mathbf{d}_2^{(0)}, \mathbf{d}_3^{(0)}, \mathbf{d}_4^{(0)}$
2. **repeat:**
3. $\mathbf{aux} = \mathbf{v_1^{(k)}} + \mathbf{d_1^{(k)}} + \mathbf{v_2^{(k)}} + \mathbf{d_2^{(k)}}$
   $\mathbf{z}^{(k+1)} \leftarrow (\mathbf{D}^T\mathbf{D} + 2\mathbf{I})^{-1} \times \left(\mathbf{aux} + \mathbf{D_h^T}(\mathbf{v_3^{(k)}} + \mathbf{d_3^{(k)}}) + \mathbf{D_v^T}(\mathbf{v_4^{(k)}} + \mathbf{d_4^{(k)}})\right)$
4. $\mathbf{v}_1^{(k+1)} \leftarrow (\mathbf{K}^T\mathbf{K} + \mu\mathbf{I})^{-1} \times \left(\mathbf{K}^T\mathbf{y} + \mu(\mathbf{z}^{(k+1)} - \mathbf{d}_1^{(k)})\right)$
5. $\mathbf{v}_2^{(k+1)} \leftarrow \max\left(0, \mathbf{z}^{(k+1)} - \mathbf{d}_2^{(k)}\right)$
6. $\mathbf{v}_3^{(k+1)} \leftarrow \mathrm{soft}\left(\mathbf{D}_h(\mathbf{z}^{(k+1)}) - \mathbf{d}_3^{(k)}, \lambda_{TV}/\mu\right)$
7. $\mathbf{v}_4^{(k+1)} \leftarrow \mathrm{soft}\left(\mathbf{D}_v(\mathbf{z}^{(k+1)}) - \mathbf{d}_4^{(k)}, \lambda_{TV}/\mu\right)$
8. **Update Lagrange multipliers:**
   $\mathbf{d}_1^{(k+1)} \leftarrow \mathbf{d}_1^{(k)} - \mathbf{z}^{(k+1)} + \mathbf{v}_1^{(k+1)}$
   $\mathbf{d}_2^{(k+1)} \leftarrow \mathbf{d}_2^{(k)} - \mathbf{z}^{(k+1)} + \mathbf{v}_2^{(k+1)}$
   $\mathbf{d}_3^{(k+1)} \leftarrow \mathbf{d}_3^{(k)} - \mathbf{D}_h\mathbf{z}^{(k+1)} + \mathbf{v}_3^{(k+1)}$
   $\mathbf{d}_4^{(k+1)} \leftarrow \mathbf{d}_4^{(k)} - \mathbf{D}_v\mathbf{z}^{(k+1)} + \mathbf{v}_4^{(k+1)}$
9. **Update iteration:** $k \leftarrow k + 1$
10. **until** $k = \mathrm{MAX\_ITERATIONS}$
11. **Reconstruction** $\hat{\mathbf{x}} = (\mathbf{I} \otimes \mathbf{E})\mathbf{z}^k$

B. CHYCA Algorithm

In order to avoid tuning the λ_{TV} parameter, another algorithm called CHYCA was proposed in [27]. In this algorithm, the reconstruction error term is constrained to $\|\mathbf{y} - \mathbf{K}\mathbf{z}\|^2 \leq \delta$ instead of being part of the objective function. The advantage of this formulation is that δ may be set with basis on the noise characteristics of the data set which are likely to be known before hand.

As in HYCA, a set of new variables per term of the objective function are introduced and the ADMM methodology [33] is used to solve the CHYCA minimization problem

$$\min_{\mathbf{z} \geq 0} \mathrm{TV}(\mathbf{z}) \quad \text{subject to :} \quad \|\mathbf{y} - \mathbf{K}\mathbf{z}\|^2 \leq \delta \tag{6}$$

where δ is a scalar value linked to the noise statistics.

Algorithm 2. Pseudocode of CHYCA algorithm

1. Initialization: set $k=0$, choose $\mu>0$, $\mathbf{E}$, $\mathbf{z^{(0)}}$, $\mathbf{v}_1^{(0)}$, $\mathbf{v}_2^{(0)}$, $\mathbf{v}_3^{(0)}$, $\mathbf{v}_4^{(0)}$, $\mathbf{v}_5^{(0)}$, $\mathbf{d}_1^{(0)}$, $\mathbf{d}_2^{(0)}$, $\mathbf{d}_3^{(0)}$, $\mathbf{d}_4^{(0)}$, $\mathbf{d}_5^{(0)}$

2. repeat:

3. $\mathbf{aux}=\mathbf{v_1^{(k)}}+\mathbf{d_1^{(k)}}+\mathbf{v_2^{(k)}}+\mathbf{d_2^{(k)}}$

$\mathbf{z}^{(k+1)}\leftarrow(\mathbf{D}^T\mathbf{D}+2\mathbf{I})^{-1}\times$

$\left(\mathbf{aux}+\mathbf{D_h^T}(\mathbf{v_3^{(k)}}+\mathbf{d_3^{(k)}})+\mathbf{D_v^T}(\mathbf{v_4^{(k)}}+\mathbf{d_4^{(k)}})\right)$

4. $\mathbf{v}_1^{(k+1)}\leftarrow(\mathbf{K}^T\mathbf{K}+\mathbf{I})^{-1}\times$

$\left[(\mathbf{z}^{(k+1)}-\mathbf{d}_1^{(k)})+\mathbf{K}^T(-\mathbf{v}_5^{(k)}+\mathbf{y}-\mathbf{d}_5^{(k)})\right]$

5. $\mathbf{v}_2^{(k+1)}\leftarrow\max\left(0,\mathbf{z}^{(k+1)}-\mathbf{d}_2^{(k)}\right)$

6. $\mathbf{v}_3^{(k+1)}\leftarrow\text{soft}\left(\mathbf{D}_h(\mathbf{z}^{(k+1)})-\mathbf{d}_3^{(k)},1/\mu\right)$

7. $\mathbf{v}_4^{(k+1)}\leftarrow\text{soft}\left(\mathbf{D}_v(\mathbf{z}^{(k+1)})-\mathbf{d}_4^{(k)},1/\mu\right)$

8. $\mathbf{y}_{\text{aux}}=\mathbf{y}-\mathbf{K}\mathbf{v}_1^{(k+1)}-\mathbf{d}_5^{(k)}$

$$\mathbf{v}_5^{(k+1)}\leftarrow\begin{cases}\mathbf{y}_{\text{aux}} & \text{if}\|\mathbf{y}_{\text{aux}}\|\le\delta\\ \frac{\delta\mathbf{y}_{\text{aux}}}{\|\mathbf{y}_{\text{aux}}\|} & \text{otherwise,}\end{cases}$$

9. Update Lagrange multipliers:

$\mathbf{d}_1^{(k+1)}\leftarrow\mathbf{d}_1^{(k)}-\mathbf{z}^{(k+1)}+\mathbf{v}_1^{(k+1)}$

$\mathbf{d}_2^{(k+1)}\leftarrow\mathbf{d}_2^{(k)}-\mathbf{z}^{(k+1)}+\mathbf{v}_2^{(k+1)}$

$\mathbf{d}_3^{(k+1)}\leftarrow\mathbf{d}_3^{(k)}-\mathbf{D}_h\mathbf{z}^{(k+1)}+\mathbf{v}_3^{(k+1)}$

$\mathbf{d}_4^{(k+1)}\leftarrow\mathbf{d}_4^{(k)}-\mathbf{D}_v\mathbf{z}^{(k+1)}+\mathbf{v}_4^{(k+1)}$

$\mathbf{d}_5^{(k+1)}\leftarrow\mathbf{d}_5^{(k)}+\mathbf{v}_5^{(k+1)}-(\mathbf{y}-\mathbf{K}\mathbf{v}_1^{(k+1)})$

10. Update iteration: $k\leftarrow k+1$

11. until $k=\text{MAX\_ITERATIONS}$

12. Reconstruction $\hat{\mathbf{x}}=(\mathbf{I}\otimes\mathbf{E})\mathbf{z}^k$

By a careful choice of the new variables, the problem is converted into a sequence of much simpler problems. With this in mind, $\iota_{B(\epsilon)}$ is defined as the indicator on a ball of radius ϵ, i.e., $\iota_{B(\epsilon)}(\mathbf{z})=0$ if $\|\mathbf{z}\|\le\varepsilon$ and $+\infty$ otherwise. With these definition in place, an equivalent way of writing the optimization problem in (6) is

$$\begin{aligned}\min_{\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3,\mathbf{v}_4,\mathbf{v}_5}\ & \phi(\mathbf{D}\mathbf{z})+\iota_{B(\delta)}(\mathbf{v}_5)+\iota_{R+}(\mathbf{v}_2)\\ \text{subject to}:\ & \mathbf{v}_1=\mathbf{z}\\ & \mathbf{v}_2=\mathbf{z}\\ & (\mathbf{v}_3,\mathbf{v}_4)=\mathbf{D}\mathbf{z}\\ & \mathbf{v}_5=\mathbf{y}-\mathbf{K}\mathbf{v}_1\end{aligned}\tag{7}$$

which we solve via ADMM in a way similar to Algorithm 1. Algorithm 2 shows the pseudocode of the CHYCA algorithm to solve the problem in (7) and how to reconstruct the data using (2).

C. Algorithmic Improvements

The step 3 of Algorithms 1 and 2 corresponds to the solution of a system of equations $\mathbf{M}\mathbf{z}^{(k+1)}=\mathbf{b}$ where $\mathbf{M}=(\mathbf{D}^T\mathbf{D}+2\mathbf{I})$ and $\mathbf{b}=\left(\mathbf{v}_1^{(k)}+\mathbf{d}_1^{(k)}+\mathbf{v}_2^{(k)}+\mathbf{d}_2^{(k)}+\mathbf{D}_h^T(\mathbf{v}_3^{(k)}+\mathbf{d}_3^{(k)})+\mathbf{D}_v^T(\mathbf{v}_4^{(k)}+\mathbf{d}_4^{(k)})\right)$. Given that the matrices $\mathbf{D}_h$ and $\mathbf{D}_v$ are block circulant, corresponding to two-dimensional (2-D) cyclic convolutions, then the computation of $\mathbf{b}$ and the solution of the linear system of equations may be implemented efficiently in the frequency domain with a complexity of $O(pn_p\log(n_p))$. However, because the complexity involved in the matrix–vector multiplications of the form $\mathbf{D}_h\mathbf{x}$ and $\mathbf{D}_v\mathbf{x}$ is of $O(pn_p)$, it may be advantageous to solve the system $\mathbf{M}\mathbf{z}=\mathbf{b}$ with a first-order stationary iterative procedure [34], which has the form

$$\begin{aligned}&\textbf{for } t=0,1,\ldots\\ &\quad\mathbf{r}_t=\mathbf{M}\mathbf{z}_t-\mathbf{b}\\ &\mathbf{z}_{t+1}=\mathbf{z}_t-\beta\mathbf{r}_t.\end{aligned}\tag{8}$$

Let $0<\lambda_{min}<\lambda_{max}$ denotes, respectively, the smallest and the largest eigenvalues of $\mathbf{M}$. Therefore, the sequence $\mathbf{z}_{t+1}$ converges to the solution of the system $\mathbf{M}\mathbf{z}=\mathbf{b}$ provided that $0<\beta<2/\lambda_{max}$ [34]. In addition, the optimal convergence factor is given by

$$\rho_{opt}=\frac{1-\lambda_{min}/\lambda_{max}}{1+\lambda_{min}\lambda_{max}}\tag{9}$$

and is obtained with $\beta_{opt}=2/(\lambda_{min}+\lambda_{max})$. For the problem in hands, we have $\lambda_{min}=2$ and $\lambda_{max}=8$ and, therefore, $\beta_{opt}=1/6$ and $\rho_{opt}=2/3$. In these conditions, the convergence rate, i.e., the number of iterations to attenuate the error $\|\mathbf{z}_t-\mathbf{z}_*\|$, where $\mathbf{z}_*=\mathbf{M}^{-1}\mathbf{b}$, by a factor of 10, is $-1/\log(2/3)=5.57$. In practice, it is not necessary to solve exactly the linear system of equations in each ADMM iteration as far as the errors are summable [33]. In ADMM iteration, we initialize (8) with $\mathbf{z}^{(k)}$ and run only a few iterations.

We now derive faster versions of HYCA and CHYCA. The modified versions of HYCA and CHYCA are termed, respectively, HYCA-FAST and CHYCA-FAST hereinafter. The pseudocode with the main modification in line 3 of Algorithms 1 and 2 is shown in Algorithm 3. In this way, the use of the fast Fourier transform to solve the $\mathbf{z}$ optimization is avoided and a fastest way to solve the optimization is provided.

Algorithm 3. Pseudocode of FAST optimization

...

3. Fast optimization:

3.1. set $\beta=1/6$ **(optimum value)**

3.2. $\mathbf{aux}=\mathbf{v_1^{(k)}}+\mathbf{d_1^{(k)}}+\mathbf{v_2^{(k)}}+\mathbf{d_2^{(k)}}$

$\mathbf{g}^{(k)}=\left(\mathbf{aux}+\mathbf{D_h^T}(\mathbf{v_3^{(k)}}+\mathbf{d_3^{(k)}})+\mathbf{D_v^T}(\mathbf{v_4^{(k)}}+\mathbf{d_4^{(k)}})\right)$

3.3. repeat:

3.3.1. $\mathbf{r}^{(f)}=2\,\mathbf{z}^{(f)}+\mathbf{D}_h^T(\mathbf{D}_h(\mathbf{z}^{(f)}))+\mathbf{D}_v^T(\mathbf{D}_v(\mathbf{z}^{(f)}))-\mathbf{g}^{(k)}$

3.3.2. $\mathbf{z}^{(f+1)}=\mathbf{z}^{(f)}-\beta*\mathbf{r}^{(f)}$

3.4. until $f=\text{DESIRED\_ITERATIONS}$

...

III. GPU Implementations

GPUs can be abstracted as an array of highly threaded streaming multiprocessors (SMs), where each multiprocessor is characterized by a single instruction multiple data (SIMD)

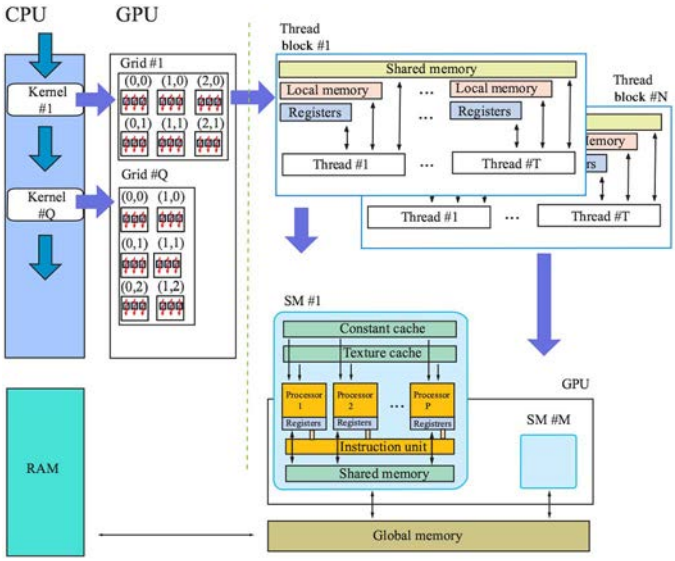

Fig. 1. Typical NVidia GPU architecture, computation, and data transfer flow from/to CPU.

architecture, i.e., in each clock cycle each processor executes the same instruction while operating on multiple data streams. Each SM has a number of streaming processors that share a control logic and instruction cache and have access to a local shared memory and to local cache memories in the multiprocessor, while the multiprocessors have access to the global GPU (device) memory. Fig. 1 presents a typical architecture and the data flow communication between CPU and GPU.

The algorithms are constructed by chaining the so-called *kernels* which operate on entire streams and which are executed by a multiprocessor, taking one or more streams as inputs and producing one or more streams as outputs. Thereby, data-level parallelism is exposed to hardware, and kernels can be concurrently applied without any sort of synchronization. The kernels can perform a kind of batch processing arranged in the form of a grid of blocks where each block is composed by a group of threads that share data efficiently through the shared local memory and synchronize their execution for coordinating accesses to memory. As a result, there are different levels of memory in the GPU for the thread, block, and grid concepts. There is also a maximum number of threads that a block can contain (depending on the GPU model), however, the number of threads that can be concurrently executed is much larger due to the fact that several blocks executed by the same kernel can be managed concurrently. With the above ideas in mind, the proposed implementations for HYCA, CHYCA, and their fast optimizations are detailed as follows.

A. P-HYCA Implementation and Its Fast Optimization

The implementation of P-HYCA algorithm starts with an initialization step. The cuBLAS and cuFFT libraries are first initialized. After that the compressed hyperspectral image is loaded band by band from the hard disk to the main memory of the GPU. This arrangement intends to access consecutive pixels in the same wavelength in parallel by the processing kernels (coalesced accesses to memory). This means that the ith thread of a block will access the ith pixel for a given wavelength. This technique is used to maximize global memory bandwidth and minimize the number of bus transactions.

Once the original image is loaded in the global memory, the kernel `Compute_compression` performs the projections between the random vectors and the image pixels in order to compress the data. For this purpose each thread will compute the multiplication of the matrix $\mathbf{H}_i$ with its corresponding pixel, so that the total number of threads will be equal to the number of pixels in the data set. The grid configuration of this kernel in the GPU will contain Num_b blocks with the maximum number of threads supported by the architecture (1024 for the GPU considered). Thus, the number of blocks Num_b will be given by the expression

$$Num_b = \left\lceil \frac{n_p}{1024} \right\rceil. \tag{10}$$

Due to the fact that we have a small set of $\mathbf{H}_i$ matrices which are systematically multiplied by the image pixels located inside spatial windows, the compression matrices $\mathbf{H}_i$ are stored in shared memory in order to optimize the memory access to the $\mathbf{H}_i$ matrices. At the end of this process, the threads will store the compressed measurements $\mathbf{y}$ in the global memory.

The next step precomputes the fixed terms $(\mathbf{K}^T\mathbf{K}+\mu\mathbf{I})^{-1}$ and $(\mathbf{K}^T\mathbf{y})$ of Algorithm 1 in order to avoid repeated computations inside the main loop from lines 2 to 10. Herein, the term $(\mathbf{K}^T\mathbf{K}+\mu\mathbf{I})^{-1}$ in line 4 of Algorithm 1 is computed in the CPU using the LAPACK[3] package due to the fact that the size of these matrices is small and it is not worth to perform this computation in the GPU. However, $(\mathbf{K}^T\mathbf{y})$ is computed using a kernel called `Compute_KTY`, which will perform the multiplication of the matrix $\mathbf{K}^T$ by its corresponding pixel. The grid configuration for this kernel is given by expression (10) as explained before. In this kernel, the matrix $\mathbf{K}$ is stored in shared memory to optimize the memory access to the elements of this matrix. It is important to emphasize that we have declared shared memory dynamically in all kernel launch configurations. In this case, the shared memory allocation (size per thread block) must be specified (in bytes) using an optional third execution configuration parameter. Inside the kernel, the shared memory array is declared by means of an unsized extern array syntax, `__extern__ double s[]`. The size is simply determined from the third execution configuration parameter when the kernel is launched.

The optimization of $\mathbf{z}$ in line 3 of Algorithm 1 is carried out in two steps. First, a kernel computes the right side of the equation: $\mathbf{v}_1^{(k)}+\mathbf{d}_1^{(k)}+\mathbf{v}_2^{(k)}+\mathbf{d}_2^{(k)}+\mathbf{D}_h^T(\mathbf{v}_3^{(k)}+\mathbf{d}_3^{(k)})+\mathbf{D}_v^T(\mathbf{v}_4^{(k)}+\mathbf{d}_4^{(k)})$; here each thread computes one element and the grid configuration is the same than the previous kernels. Later, the optimization with respect to $\mathbf{z}$ is performed using the cuFFT. Herein, two Fourier transform types were used: `real-to-complex` (R2C) and `complex-to-real` (C2R). Finally, the result is stored in global memory.

The optimization of $\mathbf{v}_1$ in line 4 of Algorithm 1 is carried out with a kernel called `Optimize_v1`. This kernel uses the same grid configuration as the previous ones. This kernel also makes

[3][Online]. Available: http://www.netlib.org/lapack/.

use of the shared memory to optimize the memory access to the matrix $(\mathbf{K}^T\mathbf{K}+\mu\mathbf{I})^{-1}$, which was precomputed before. The resulting $\mathbf{v}_1$ is stored in the global memory.

Line 5 in Algorithm 1 is carried out with a kernel called `Optimize_v2`, which computes the maximum between 0 and $\mathbf{z}^{(k+1)}-\mathbf{d}_2^{(k)}$. This kernel uses as many threads as the number of elements of the vector (n_p), with the same grid configuration as the previous kernels.

The optimization of $\mathbf{v}_3$ is carried out jointly by a single kernel called `Optimize_v3_v4`, which computes the lines 6 and 7 in Algorithm 1. This kernel uses the same configuration as the previous kernels with as many threads as the image pixels. In this kernel each thread computes the horizontal and vertical differences of one element and performs the soft function for the corresponding element, where $\mathrm{soft}(\cdot,\tau)$ denotes the application of the soft-threshold function $\mathbf{b}\mapsto\mathbf{b}\frac{\max\{\|\mathbf{b}\|_2-\tau,0\}}{\max\{\|\mathbf{b}\|_2-\tau,0\}+\tau}$.

Finally, Lagrange multipliers update is computed with two kernels called `Compute_d12` and `Compute_d34` which, respectively, compute the update of the variables $\mathbf{d}_1,\mathbf{d}_2$ and $\mathbf{d}_3,\mathbf{d}_4$.

The algorithm repeats this process until a number of iterations k is reached. Once the estimated fractional abundances $\mathbf{z}$ are computed, the algorithm reconstructs the original hyperspectral data set multiplying by the endmember matrix. This process is performed using the `cuBLAS` library. Specifically, the cublasSgemm function of cuBLAS was used to reconstruct the image $\hat{\mathbf{x}}$.

Finally, a fast optimized version termed P-HYCA-FAST has been implemented following the previous strategies except the implementation of the line 3 from Algorithm 1. In this version, a new kernel called `Compute_z_Fast` is used to compute lines 3.3.1 and 3.3.2 of Algorithm 3. The kernel uses the same grid configuration as the previous ones (one thread per pixel). This kernel is called f times and performs the update of both $\mathbf{r}$ and $\mathbf{z}$ variables. This kernel optimizes the memory access by using the L1 cache memory. On CUDA devices with a compute capability 2.0 or later (our case), both the L1 cache and shared memory use the same hardware resources, but CUDA allows setting a preferred size for the L1 cache memory. The device sets a preference of larger L1 cache and smaller shared memory using the CUDA directive `cudaFuncSetCacheConfig("name_kernel," cudaFuncCachePreferL1)`. Due to the fact that this kernel does not use shared memory, the memory access can be optimized with a larger L1 cache memory in order to increase the kernel performance.

B. P-CHYCA Implementation and Its Fast Optimization

The proposed P-CHYCA method follows the same strategy as in the implementation of previous methods. The main differences with regards to Algorithm 1 are highlighted below. Line 4 in Algorithm 2 is carried out with two kernels called `Compute_KT_VYD` and `Optimize_v1_CHYCA`, respectively. The grid configuration for the first kernel is given by expression (10) which calculates the operation: $\mathbf{K}^T(-\mathbf{v}_5^{(k+1)}+\mathbf{y}-\mathbf{d}_5^{(k)})$, where the matrix $\mathbf{K}$ is stored in shared memory to optimize the memory access to the elements of this matrix. In first place, each thread in this kernel computes one element of the operation: $(-\mathbf{v}_5^{(k+1)}+\mathbf{y}-\mathbf{d}_5^{(k)})$, storing the resulting $\mathbf{VYD}$ structure in global memory. Then, each thread computes one element in the matrix multiplication between $\mathbf{K}^T$ and $\mathbf{VYD}$. Once the above operation is completed, the second kernel is executed to compute the matrix multiplication between $(\mathbf{K}^T\mathbf{K}+\mu\mathbf{I})^{-1}$, stored in shared memory, and the right side of the equation in Line 4, where the same grid configuration as the previous one is used. Finally, the optimization of $\mathbf{v}_1$ is stored in global memory.

On the other hand, the optimization of $\mathbf{v}_5$ is divided into two kernels in Line 8. The first one is termed `Compute_Y_KV1_D5`, here the grid configuration is the same than the previous kernels. Thereby, each thread computes one element of $\mathbf{y}-\mathbf{d}_5$ and subtracts the matrix multiplication $\mathbf{K}$ by its corresponding element in $\mathbf{v}_1^{(k+1)}$ leading to the $\mathbf{Y\_KV1\_D5}$ structure in global memory. Due to the fact that this kernel does not use shared memory, the memory accesses can be optimized with a larger L1 cache memory in order to increase the kernel performance. Later the second kernel performs a reduction process with a grid configuration given by expression (11). This process uses shared memory and coalesced memory accesses to add each element in $\mathbf{Y\_KV1\_D5}$ structure. Note that in this case, the number of threads per block is reduced to 512 in order to get a better performance

$$Num_b=\left\lceil\frac{n_p\times q}{512}\right\rceil. \tag{11}$$

Once the optimization $\mathbf{v}_5$ is completed and stored in global memory, the Lagrange multipliers are updated following the P-HYCA scheme and the reconstruction of the original hyperspectral data set are performed. Note that the $\mathbf{y}-\mathbf{K}\mathbf{v}_1^{(k+1)}$ operation has been calculated previously.

Finally, a fast optimized version termed P-CHYCA-FAST has been implemented as shown in Algorithm 3 using the same strategy as in P-HYCA-FAST implementation.

IV. Experimental Results

A. Hyperspectral Image Data

The experiments are carried out using three hyperspectral images. The first two synthetic data sets used in our experiments, denoted hereinafter as synthetic_1 and synthetic_2 were generated from spectral signatures randomly selected from the U.S. Geological Survey (USGS).[4] Synthetic_1 consists of a set of 5×5 squares of 10×10 pixels each one, yielding a total size of 110×110 pixels (10.8 MB). The first row of squares contains the endmembers, the second row contains mixtures of two endmembers, the third row contains mixtures of three endmembers, and so on. Fig. 2 displays the ground-truth abundance maps used for generating the simulated imagery. Synthetic_2 consists of a set of 30 signatures from the USGS library and it is generated using the procedure described in [35] to simulate natural spatial patterns, composed by a total size of

[4][Online]. Available: http://speclab.cr.usgs.gov.

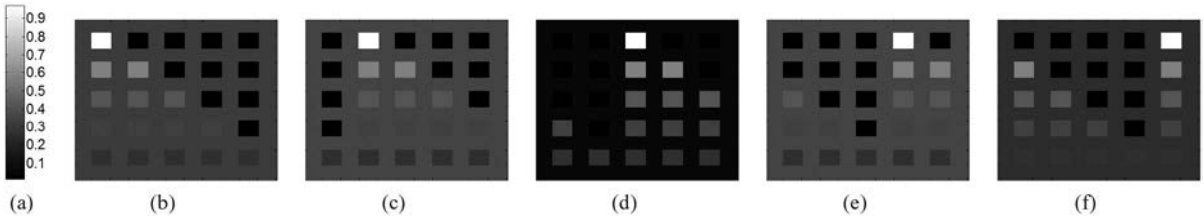

Fig. 2. Ground-truth abundance maps of endmembers in the synthetic_1 hyperspectral data represented in gray scale. (a) Gray scale bar (white and black represent 1 and 0, respectively), endmembers (b) #1, (c) #2, (d) #3, (e) #4, and (f) #5.

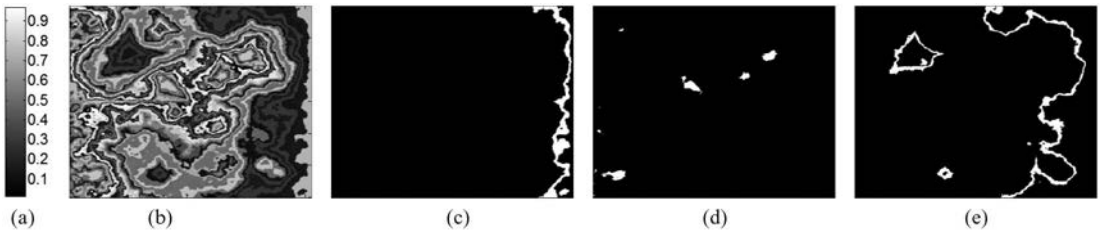

Fig. 3. Gray-scale composition and three examples of ground-truth abundance maps of endmembers in the synthetic_2 hyperspectral data. (a) Gray scale bar (white and black represent 1 and 0, respectively), (b) gray-scale composition, endmembers (c) #5, (d) #23, (e) #29.

600×512 pixels (275 MB). Fig. 3 displays a gray-scale composition and three examples of ground-truth abundance maps from the simulated image.

The third hyperspectral image considered in experiments is the well-known AVIRIS Cuprite scene, available online in reflectance units after atmospheric correction.[5] This scene has been widely used to validate the performance of endmember extraction algorithms. The portion used in experiments corresponds to a 250×190 pixels subset of the sector labeled as f970619t01p02_r02_sc03.a.rfl in the online data, which comprises 188 spectral bands in the range from 400 to 2500 nm and a total size of around 36 MB. Water absorption bands as well as bands with low signal-to-noise ratio (SNR) were removed prior to the analysis. The site is well understood mineralogically, and has several exposed minerals of interest, including *Alunite*, *Buddingtonite*, *Calcite*, *Kaolinite*, and *Muscovite*. For this subimage, the number of endmembers were estimated by Hysime method [28] and their signatures were extracted in a very fast way through VCA algorithm [29].

B. Analysis of Accuracy

In order to evaluate the accuracy of the proposed implementations in terms of reconstruction error, the normalized mean squared error (NMSE) between the original image and the reconstructed image (after data compression and decompression) and the peak signal-to-noise ratio (PSNR) have been adopted as performance indicators. The first one is given by

$$\mathrm{NMSE} = \|\widehat{\mathbf{x}} - \mathbf{x}\|_F^2 / \|\mathbf{x}\|_F^2 \tag{12}$$

where $\mathbf{x}$ and $\widehat{\mathbf{x}}$ denote the original and reconstructed hyperspectral images, respectively, and $\|.\|_F$ denotes the Frobenius norm. The second performance indicator is given by

$$\mathrm{PSNR} = 10 \log_{10} \frac{\max(\mathbf{x})^2 \times n_p \times n_b}{\|\widehat{\mathbf{x}} - \mathbf{x}\|_F^2}. \tag{13}$$

[5][Online]. Available: http://aviris.jpl.nasa.gov.

With the aim of evaluating the proposed implementations at different levels of compression we defined the number of measurements $q = p = 5$ in the case of the synthetic_1 data set, $q = p/2 = 15$ in the case of synthetic_2, and finally $q = 5 \ll p$ for the real Cuprite data set. Thus, the compression ratios (n_b/q) are 44.8, 14.93, and 37.60. The window sizes considered were $ws = 2 \times 2$, 4×4, and 2×2, respectively. These parameter values were defined empirically.

Fig. 4 shows the PSNR achieved for the proposed implementations as a function of the number of iterations. In this experiment one can see that P-HYCA or P-CHYCA and their fast versions provide very similar values when the number of iterations is high. However, in the first few iterations we can appreciate the higher performance of the P-HYCA and P-CHYCA versions. On the other hand, P-CHYCA presents highest SNR in the AVIRIS Cuprite scene demonstrating their good performance in real data sets.

Fig. 5 presents the NMSE maps for each data set for the proposed implementations with $k = 175$ iterations. Notice that the proposed methods provide very good results with very low NMSE values. Note also that the scale of these figures is the same in order to compare the P-HYCA or P-CHYCA and their fast optimizations for each considered data set.

C. Analysis of Parallel Performance

The proposed parallel versions have been tested on two different GPUs:

1) The first GPU (denoted hereinafter as GPU1) is the NVidia GeForce GTX 590,[6] which features 1024 processor cores operating at 1.215 GHz, total dedicated memory of 3072 MB (1536 MB by GPU), 1707 MHz memory clock (with 384-bits GDDR5 interface per GPU) and memory bandwidth of 327.7 GB/s. This GPU is

[6][Online]. Available: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-590/specifications.

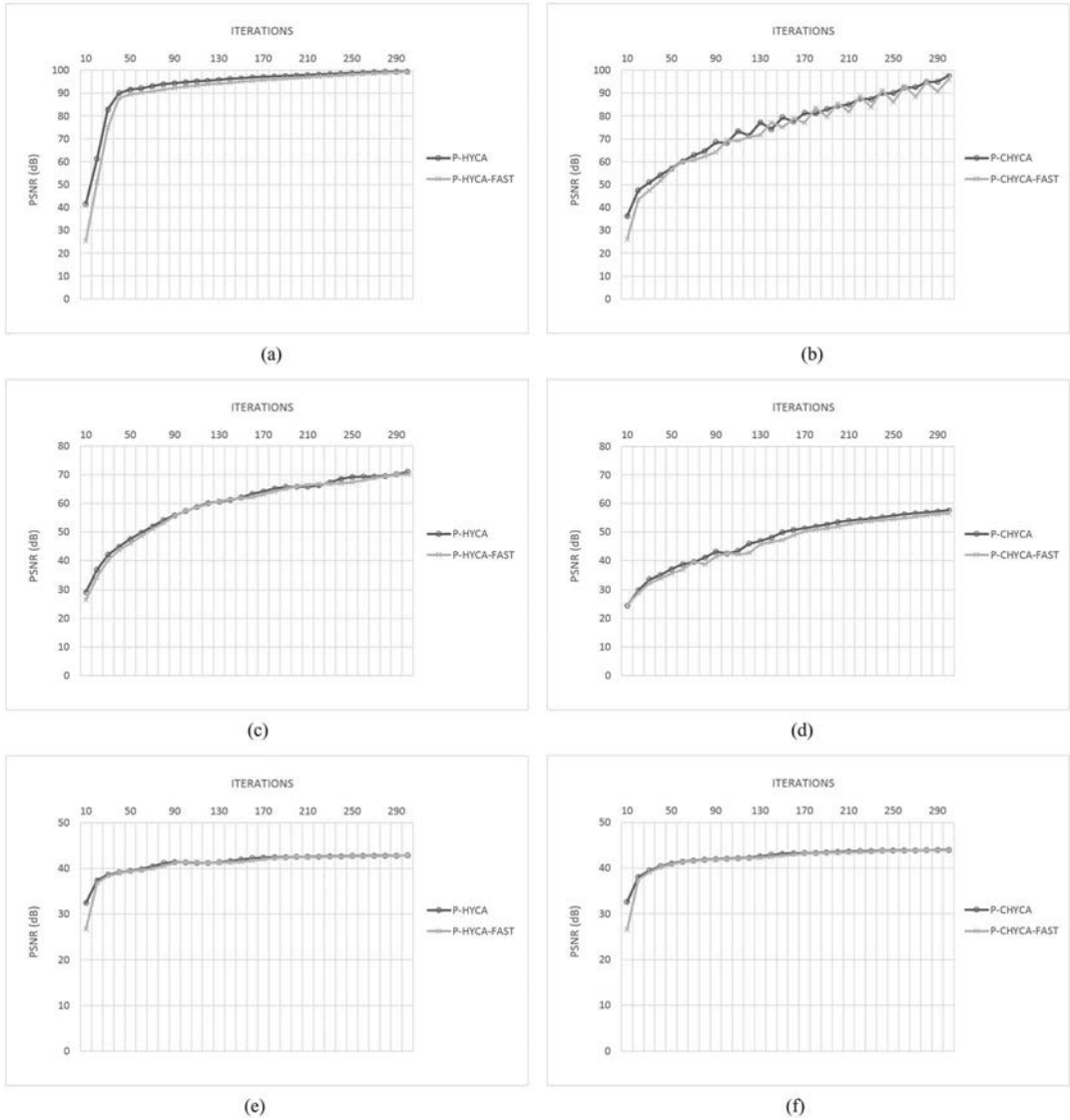

Fig. 4. PSNR between the original and the reconstructed images for (a), (c), and (e) P-HYCA and P-HYCA-FAST methods and (b), (d), and (f) P-CHYCA and P-CHYCA-FAST methods. (a) Synthetic_1 for values of $q = 5$ and $ws = 4$. (b) Synthetic_1 for values of $q = 5$ and $ws = 4$. (c) Synthetic_2 for values of $q = 15$ and $ws = 16$. (d) Synthetic_2 for values of $q = 15$ and $ws = 16$. (e) AVIRIS Cuprite for values of $q = 5$ and $ws = 4$. (f) AVIRIS Cuprite for values of $q = 5$ and $ws = 4$.

connected to a multicore Intel i7-2600 CPU at 3.40 GHz with four physical cores and 16 GB of DDR3 RAM memory.

2) The second GPU (denoted hereinafter as GPU2) used is the NVidia GeForce GTX TITAN,[7] which features 2688 processor cores operating at 876 MHz, total dedicated memory of 6144 MB, 6.0 Gbps memory clock (with 384-bit GDDR5 interface) and memory bandwidth of 288.4 GB/s. This GPU is connected to a multicore Intel i7-4770K CPU at 3.50 GHz with four physical cores and 32 GB of DDR3 RAM memory.

Before describing the parallel algorithm performance results, it is important to emphasize that the GPU versions provide exactly the same results as the serial versions of the implemented algorithms, using the gcc-4.8.2 (gnu compiler default) with optimization flags -O3 (for the single-core version) to exploit data locality and avoid redundant computations. Note that for the serial HYCA implementation, the 3.3.4 FFTW library[8] version has been used. Hence, the only difference between the serial and parallel versions is the time they need to complete their calculations. The serial algorithms were executed in one of the available CPU cores, whereas the parallel version developed for the GPU GTX 590 has been executed using one of the two GPUs available in the system. For each experiment, 10 runs were performed and the mean values were reported (these times were always very similar, with differences on the order of a few milliseconds).

Table I shows the execution time of the compression method [see expression (1)] for the synthetic and real data sets with the compression ratios shown in Section IV-B. These results

[7][Online]. Available: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan/specifications.

[8][Online]. Available: http://www.fftw.org/#documentation.

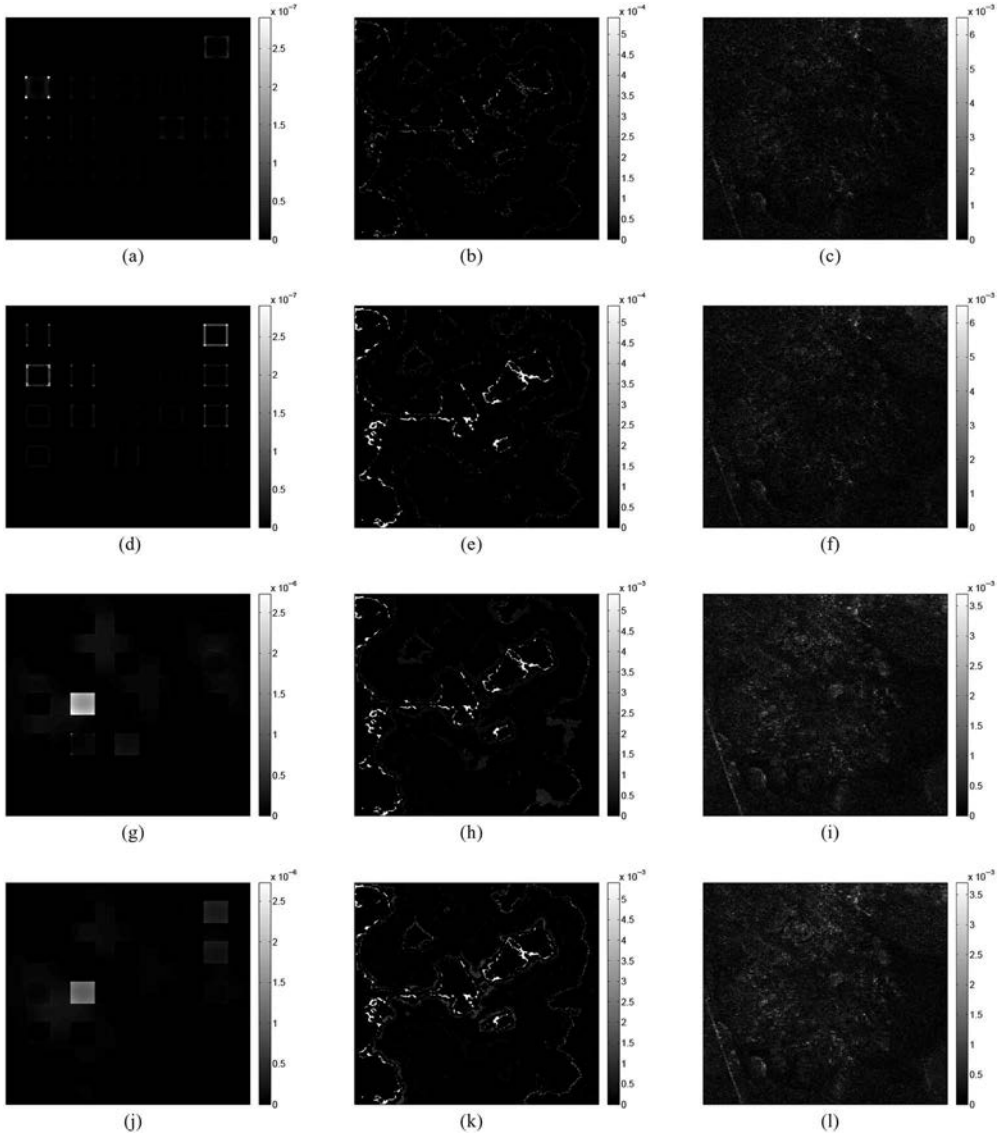

Fig. 5. Each column represents the NMSE maps between the original (Synthetic_1, Synthetic_2, and AVIRIS Cuprite, respectively) and the reconstructed data sets for different compression ratios. (a)–(c) P-HYCA, (d)–(f) P-HYCA-FAST, (g)–(i) P-CHYCA, and (j)–(l) P-CHYCA-FAST algorithms. (a) $q = 5$ and $ws = 4$. (b) $q = 15$ and $ws = 16$. (c) $q = 5$ and $ws = 4$. (d) $q = 5$ and $ws = 4$. (e) $q = 15$ and $ws = 16$. (f) $q = 5$ and $ws = 4$. (g) $q = 5$ and $ws = 4$. (h) $q = 15$ and $ws = 16$. (i) $q = 5$ and $ws = 4$. (j) $q = 5$ and $ws = 4$. (k) $q = 15$ and $ws = 16$. (l) $q = 5$ and $ws = 4$.

report speedups higher than $56\times$, achieved on the GPU2 device. As expected the speedup is higher for the largest image (Synthetic_2), this is manly because the parallelization is performed in a pixel-based scheme, thus the speedup grows with the number of pixels of the image. Notice that the speedup results for GPU2 are better than GPU1 since GPU2 has more processing cores and more dedicated memory on the device.

Tables II–V summarize the time results and speedups measured after processing synthetic and real hyperspectral images on the considered GPU platforms during 175 iterations. The processing times for the P-HYCA method and P-HYCA-FAST methods are presented in Tables II and III, respectively. It is worth noting that P-HYCA-FAST achieves higher speedups factors when compared with P-HYCA method. After comparing P-CHYCA and P-CHYCA-FAST processing times, which are presented in Tables IV and V, one can conclude that P-CHYCA-FAST achieves better performance.

It should be also noted that the cross-track line scan time in AVIRIS, a push-broom instrument [2], is quite fast (8.3 ms to collect 512 full pixel vectors). This introduces the need to process the considered scenes in less than 0.196, 0.770, and 4.980 s, for Synthetic_1, AVIRIS Cuprite, and Synthetic_2 datasets, respectively, in order to achieve real-time performance. As shown in Tables II–V, all the scenes could be processed in real-time using method P-HYCA-FAST on GPU2 device, which results in speedups higher than $100\times$, while all other methods only could be processed in near real time. This is due to the optimizations that include the iterative method in order to calculate the $\mathbf{z}$ coefficients, being one of the most time consuming part for both parallel implementations.

TABLE I
PROCESSING TIMES (IN SECONDS) AND SPEEDUPS ACHIEVED FOR THE PROPOSED COMPRESSION METHODOLOGY IN TWO DIFFERENT PLATFORMS AND TESTED WITH SYNTHETIC AND REAL DATA SETS

	Synthetic _1			AVIRIS Cuprite			Synthetic _2		
	CPU	GPU1	GPU2	CPU	GPU1	GPU2	CPU	GPU1	GPU2
RAM $\rightarrow GlobalMem.$	–	0.0034	0.0022	–	0.0109	0.0064	–	0.0819	0.0455
Compute compression	0.0127	0.0007	0.0009	0.0474	0.0018	0.0026	5.9926	0.0417	0.0521
RAM $\leftarrow GlobalMem.$	–	0.0003	0.0002	–	0.0010	0.0008	–	0.0112	0.0090
Total time	0.0127	0.0045	0.0034	0.0474	0.0138	0.0097	5.9926	0.1348	0.1066
Speedup (CPU time / GPU time)	–	2.82×	3.76×	–	3.45×	4.89×	–	44.46×	56.24×

TABLE II
PROCESSING TIMES (IN SECONDS) AND SPEEDUPS ACHIEVED FOR THE P-HYCA IN TWO DIFFERENT PLATFORMS AND TESTED WITH SYNTHETIC AND REAL DATA SETS

	Synthetic _1			AVIRIS Cuprite			Synthetic _2		
	CPU	GPU1	GPU2	CPU	GPU1	GPU2	CPU	GPU1	GPU2
RAM $\rightarrow GlobalMem.$	–	0.0002	0.0001	–	0.0007	0.0004	–	0.0112	0.0063
$(\mathbf{K^T K} + \mu\mathbf{I})^{-1}$	0.0001	0.0001	0.0001	0.0001	0.0001	0.0002	0.0019	0.0019	0.0020
RAM $\rightarrow GlobalMem.$	–	≈ 0.0000	≈ 0.0000	–	≈ 0.0000	≈ 0.0000	–	≈ 0.0000	≈ 0.0000
$(\mathbf{K}^T\mathbf{y})$	0.0004	0.0000	0.0000	0.0050	0.0003	0.0002	0.2676	0.0089	0.0053
Compute $\mathbf{z}$ (line 3)	0.3407	0.0849	0.0922	5.7367	0.4354	0.3537	129.1343	2.3902	1.4262
Compute $\mathbf{v}_1$ (line 4)	0.0818	0.0093	0.0068	2.9370	0.2055	0.1348	99.8648	3.2456	1.9940
Compute $\mathbf{v}_2$ (line 5)	0.0085	0.0032	0.0031	0.2434	0.0231	0.0159	2.4120	0.2317	0.1433
Compute $\mathbf{v}_3$, $\mathbf{v}_4$ (lines 6–7)	0.0540	0.0074	0.0064	1.9304	0.0620	0.0622	30.8119	0.6336	0.5584
Compute $\mathbf{d}$ (line 8)	0.0637	0.0121	0.0093	2.1348	0.1342	0.0808	72.6970	1.4802	0.8704
Reconstruction (line 11)	0.0103	0.0008	0.0004	0.1149	0.0015	0.0007	1.6387	0.0101	0.0052
RAM $\leftarrow GlobalMem.$	–	0.0036	0.0029	–	0.0110	0.0087	–	0.0805	0.0639
Total time	0.5594	0.1219	0.1213	13.1022	0.8738	0.6575	336.8282	8.0937	5.0750
Speedup (CPU time / GPU time)	–	4.59×	4.61×	–	15.00×	19.93×	–	41.62×	66.37×

TABLE III
PROCESSING TIMES (IN SECONDS) AND SPEEDUPS ACHIEVED FOR THE P-HYCA-FAST IN TWO DIFFERENT PLATFORMS AND TESTED WITH SYNTHETIC AND REAL DATA SETS

	Synthetic _1			AVIRIS Cuprite			Synthetic _2		
	CPU	GPU1	GPU2	CPU	GPU1	GPU2	CPU	GPU1	GPU2
RAM $\rightarrow GlobalMem.$	–	0.0002	0.0001	–	0.0007	0.0004	–	0.0112	0.0063
$(\mathbf{K^T K} + \mu\mathbf{I})^{-1}$	0.0001	0.0001	0.0001	0.0001	0.0001	0.0002	0.0018	0.0019	0.0020
RAM $\rightarrow GlobalMem.$	–	≈ 0.0000	≈ 0.0000	–	≈ 0.0000	≈ 0.0000	–	≈ 0.0000	≈ 0.0000
$(\mathbf{K}^T\mathbf{y})$	0.0004	0.0000	0.0000	0.0050	0.0003	0.0002	0.2670	0.0089	0.0056
Compute $\mathbf{z}$ (line 3)	0.2438	0.0196	0.0168	7.6454	0.1720	0.1402	287.2686	1.7839	1.2753
Compute $\mathbf{v}_1$ (line 4)	0.0541	0.0081	0.0062	2.5694	0.2009	0.1327	94.8091	3.1997	1.9570
Compute $\mathbf{v}_2$ (line 5)	0.0084	0.0032	0.0029	0.2448	0.0231	0.0158	2.4095	0.2315	0.1432
Compute $\mathbf{v}_3$, $\mathbf{v}_4$ (lines 6–7)	0.0536	0.0075	0.0063	1.8909	0.0613	0.0616	33.5687	0.6355	0.5783
Compute $\mathbf{d}$ (line 8)	0.0642	0.0113	0.0091	2.1797	0.1261	0.0807	73.1620	1.3492	0.8690
Reconstruction (line 11)	0.0099	0.0008	0.0005	0.1155	0.0030	0.0008	1.6378	0.0250	0.0053
RAM $\leftarrow GlobalMem.$	–	0.0036	0.0030	–	0.0109	0.0088	–	0.0806	0.0638
Total time	0.4344	0.0544	0.0450	14.6508	0.5985	0.4414	493.1245	7.3274	4.9059
Speedup (CPU time / GPU time)	–	7.98×	9.66×	–	24.48×	33.19×	–	67.30×	100.52×

V. CONCLUSION AND FUTURE RESEARCH LINES

In this work, we have developed computationally efficient implementations of HYCA and CHYCA methods for hyperspectral CS on GPU platforms. The significant speedups reported in the experiments is expected to bridge the gap toward real-time CS of hyperspectral data sets, which is a highly desirable requirement for many remote sensing applications. The performance of the proposed implementations has been evaluated (in terms of the quality of the solutions provided and their parallel performance) using a real data set collected by

TABLE IV
PROCESSING TIMES (IN SECONDS) AND SPEEDUPS ACHIEVED FOR THE P-CHYCA IN TWO DIFFERENT PLATFORMS AND TESTED WITH SYNTHETIC AND REAL DATA SETS

	Synthetic _1			AVIRIS Cuprite			Synthetic _2		
	CPU	GPU1	GPU2	CPU	GPU1	GPU2	CPU	GPU1	GPU2
RAM $\rightarrow GlobalMem.$	–	0.0002	0.0001	–	0.0007	0.0004	–	0.0112	0.0064
$(\mathbf{K}^{\mathbf{T}}\mathbf{K}+\mu\mathbf{I})^{-1}$	0.0001	0.0001	0.0001	0.0001	0.0001	0.0002	0.0018	0.0019	0.0020
RAM $\rightarrow GlobalMem.$	–	$\approx$ 0.0000	$\approx$ 0.0000	–	$\approx$ 0.0000	$\approx$ 0.0000	–	$\approx$ 0.0000	$\approx$ 0.0000
Compute $\mathbf{z}$ (line 3)	0.3097	0.0845	0.0877	4.3824	0.4313	0.3562	43.5500	2.3433	1.4437
Compute $\mathbf{v}_1$ (line 4)	0.1841	0.0173	0.0121	4.0020	0.2630	0.1775	173.7161	4.9505	3.0786
Compute $\mathbf{v}_2$ (line 5)	0.0093	0.0033	0.0030	0.2374	0.0231	0.0157	2.4494	0.2315	0.1435
Compute $\mathbf{v}_3$, $\mathbf{v}_4$ (lines 6–7)	0.0554	0.0076	0.0061	1.8076	0.0599	0.0587	29.2135	0.6049	0.4927
Compute $\mathbf{v}_5$ (line 8)	0.0928	0.0120	0.0106	1.3509	0.0697	0.0509	93.7765	1.7767	1.1494
Compute $\mathbf{d}$ (line 9)	0.0775	0.0145	0.0115	2.2668	0.1390	0.0902	73.8639	1.5798	1.0084
Reconstruction (line 12)	0.0099	0.0008	0.0004	0.1194	0.0030	0.0007	1.6416	0.0250	0.0057
RAM $\leftarrow GlobalMem.$	–	0.0036	0.0031	–	0.0110	0.0087	–	0.0805	0.0648
Total time	0.7389	0.1439	0.1346	14.1668	1.0009	0.7592	418.2128	11.6051	7.3952
Speedup (CPU time / GPU time)	–	5.13×	5.49×	–	14.15×	18.66×	–	36.04×	56.55×

TABLE V
PROCESSING TIMES (IN SECONDS) AND SPEEDUPS ACHIEVED FOR THE P-CHYCA-FAST IN TWO DIFFERENT PLATFORMS AND TESTED WITH SYNTHETIC AND REAL DATA SETS

	Synthetic _1			AVIRIS Cuprite			Synthetic _2		
	CPU	GPU1	GPU2	CPU	GPU1	GPU2	CPU	GPU1	GPU2
RAM $\rightarrow GlobalMem.$	–	0.0002	0.0001	–	0.0007	0.0004	–	0.0112	0.0068
$(\mathbf{K}^{\mathbf{T}}\mathbf{K}+\mu\mathbf{I})^{-1}$	0.0001	0.0001	0.0001	0.0001	0.0001	0.0002	0.0018	0.0019	0.0020
RAM $\rightarrow GlobalMem.$	–	$\approx$ 0.0000	$\approx$ 0.0000	–	$\approx$ 0.0000	$\approx$ 0.0000	–	$\approx$ 0.0000	$\approx$ 0.0000
Compute $\mathbf{z}$ (line 3)	0.2098	0.0196	0.0159	6.4152	0.1720	0.1404	207.3609	1.7864	1.2753
Compute $\mathbf{v}_1$ (line 4)	0.1663	0.0153	0.0116	3.8173	0.2491	0.1756	171.9128	4.9197	3.0257
Compute $\mathbf{v}_2$ (line 5)	0.0091	0.0032	0.0029	0.2376	0.0231	0.0156	2.4638	0.2315	0.1437
Compute $\mathbf{v}_3$, $\mathbf{v}_4$ (lines 6 –7)	0.0537	0.0076	0.0060	1.7976	0.0597	0.0583	29.1566	0.6046	0.4843
Compute $\mathbf{v}_5$ (line 8)	0.0928	0.0120	0.0103	1.3480	0.0695	0.0507	93.1205	1.7768	1.1439
Compute $\mathbf{d}$ (line 9)	0.0778	0.0145	0.0114	2.2669	0.1389	0.0901	73.9416	1.5783	1.0142
Reconstruction (line 12)	0.0098	0.0008	0.0005	0.1205	0.0030	0.0008	1.6389	0.0250	0.0054
RAM $\leftarrow GlobalMem.$	–	0.0036	0.0030	–	0.0110	0.0087	–	0.0806	0.0700
Total time	0.6194	0.0770	0.0617	16.0033	0.7272	0.5408	579.5970	11.0159	7.1714
Speedup (CPU time / GPU time)	–	8.05×	10.03×	–	22.01×	29.59×	–	52.61×	80.82×

the AVIRIS instrument and synthetic scenarios. The experimental results reported in this paper indicate that remotely sensed hyperspectral imaging can greatly benefit from the development of efficient implementations of CS algorithms in specialized hardware devices for better exploitation of high-dimensional data sets. In this case, real-time performance could be obtained using the P-HYCA-FAST version and the NVidia GeForce GTX TITAN device, one of the latest GPU models characterized by the integration of a significant number of processing cores.

Although the results reported in this paper are very encouraging, in future work we will continue exploring additional strategies for optimization, such as splitting the original hyperspectral image into subimages and applying a multiGPU implementation to each of them. We are also investigating the use of OpenCL as a computing standard for multicore architectures. Moreover, other high-performance computing architectures such as digital signal processors (DSPs) or FPGAs will be also explored due to their capacity to be used as onboard processing modules in airborne and, particularly, spaceborne Earth observation missions.

REFERENCES

[1] J. M. Bioucas-Dias, A. Plaza, G. Camps-Valls, P. Scheunders, N. Nasrabadi, and J. Chanussot, "Hyperspectral remote sensing data analysis and future challenges," *IEEE Geosci. Remote Sens. Mag.*, vol. 1, no. 2, pp. 6–36, Jun. 2013.

[2] R. O. Green *et al.*, "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS)," *Remote Sens. Environ.*, vol. 65, no. 3, pp. 227–248, 1998.

[3] A. Plaza and C.-I. Chang, *High Performance Computing in Remote Sensing*. New York, NY, USA: Taylor & Francis, 2007.
[4] J. M. Bioucas-Dias, A. Plaza, N. Dobigeon, M. Parente, Q. Du, P. Gader, and J. Chanussot, "Hyperspectral unmixing: Geometrical, statistical, and sparse regression-based approaches," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 5, no. 2, pp. 354–379, Apr. 2012.
[5] Y. Altmann, N. Dobigeon, and J.-Y. Tourneret, "Unsupervised post-nonlinear unmixing of hyperspectral images using a hamiltonian monte carlo algorithm," *IEEE Trans. Image Process.*, vol. 23, no. 6, pp. 2663–2675, Jun. 2014.
[6] A. Plaza, P. Martinez, R. Perez, and J. Plaza, "A quantitative and comparative analysis of endmember extraction algorithms from hyperspectral data," *IEEE Trans. Geosci. Remote Sens.*, vol. 42, no. 3, pp. 650–663, Mar. 2004.
[7] A. Plaza, D. Valencia, J. Plaza, and P. Martinez, "Commodity cluster-based parallel processing of hyperspectral Imagery," *J. Parallel Distrib. Comput.*, vol. 66, no. 3, pp. 345–358, 2006.
[8] A. Plaza and C.-I. Chang, "Clusters versus FPGA for parallel processing of hyperspectral imagery," *Int. J. High Perform. Comput. Appl.*, vol. 22, no. 4, pp. 366–385, 2008.
[9] C. Gonzalez, D. Mozos, J. Resano, and A. Plaza, "FPGA implementation of the N-FINDR algorithm for remotely sensed hyperspectral image analysis," *IEEE Trans. Geosci. Remote Sens.*, vol. 50, no. 2, pp. 374–388, Feb. 2012.
[10] S. Bernabe, S. Sanchez, A. Plaza, S. Lopez, J. A. Benediktsson, and R. Sarmiento, "Hyperspectral unmixing on GPUs and multi-core processors: A comparison," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 6, no. 3, pp. 1386–1398, Jun. 2013.
[11] S. Bernabe, S. Lopez, A. Plaza, and R. Sarmiento, "GPU implementation of an automatic target detection and classification algorithm for hyperspectral image analysis," *IEEE Geosci. Remote Sens. Lett.*, vol. 10, no. 2, pp. 221–225, Mar. 2013.
[12] A. Plaza, Q. Du, Y.-L- Chang, and R. L. King, "High performance computing for hyperspectral remote sensing," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 528–544, Sep. 2011.
[13] E. Christophe, J. Michel, and J. Inglada, "Remote sensing processing: From multicore to GPU," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 643–652, Sep. 2011.
[14] S. Sanchez, A. Paz, G. Martin, and A. Plaza, "Parallel unmixing of remotely sensed hyperspectral images on commodity graphics processing units," *Concurrency Comput. Pract. Exper.*, vol. 23, no. 13, pp. 1538–1557, 2011.
[15] A. Plaza, "Special issue on architectures and techniques for real-time processing of remotely sensed images," *J. Real-Time Image Process.*, vol. 4, no. 3, pp. 191–193, 2009.
[16] G. Motta, F. Rizzo, and J. A. Storer, *Hyperspectral Data Compression*. New York, NY, USA: Springer, 2006.
[17] B. Huang, *Satellite Data Compression*. New York, NY, USA: Springer, 2011.
[18] C. Li, T. Sun, K. Kelly, and Y. Zhang, "A compressive sensing and unmixing scheme for hyperspectral data processing," *IEEE Trans. Image Process.*, vol. 21, no. 3, pp. 1200–1210, Mar. 2012.
[19] Q. Zhang, R. Plemmons, D. Kittle, D. Brady, and S. Prasad, "Joint segmentation and reconstruction of hyperspectral data with compressed measurements," *Appl. Opt.*, vol. 50, no. 22, pp. 4417–4435, 2011.
[20] M. Golbabaee and P. Vandergheynst, "Joint trace/TV norm minimization: A new efficient approach for spectral compressive imaging," in *Proc. IEEE Int. Conf. Image Process. (ICIP'12)*, Sep. 2012, pp. 933–936, doi: 10.1109/ICIP.2012.6467014.
[21] P. V. M. Golbabaee and S. Arberet, "Compressive source separation: Theory and methods for hyperspectral imaging," arXiv preprint arXiv:1208.4505, 2012.
[22] Q. Du and J. E. Fowler, "Low-complexity principal component analysis for hyperspectral image compression," *Int. J. High Perform. Comput. Appl.*, vol. 22, pp. 273–286, 2009.
[23] C. Song, Y. Li, and B. Huang, "A GPU-accelerated wavelet decompression system with SPIHT and Reed-Solomon decoding for satellite images," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 683–690, Sep. 2011.
[24] S.-C. Wei and B. Huang, "GPU acceleration of predictive partitioned vector quantization for ultraspectral sounder data compression," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 677–682, Sep. 2011.
[25] D. Donoho, M. Elad, and V. Temlyakov, "Stable recovery of sparse overcomplete representations in the presence of noise," *IEEE Trans. Inf. Theory*, vol. 52, no. 1, pp. 6–18, Jan. 2006.
[26] J. E. Candès, T. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *Commun. Pure Appl. Math.*, vol. 59, no. 8, p. 1207, 2006.
[27] G. Martin, J. M. Bioucas-Dias, and A. Plaza, "HYCA: A new technique for hyperspectral copressive sensing," *IEEE Trans. Geosci. Remote Sens.*, vol. 53, no. 5, pp. 2819–2831, May 2015.
[28] J. M. Bioucas-Dias and J. M. P. Nascimento, "Hyperspectral subspace identification," *IEEE Trans. Geosci. Remote Sens.*, vol. 46, no. 8, pp. 2435–2445, Aug. 2008.
[29] J. M. P. Nascimento and J. M. Bioucas-Dias, "Vertex component analysis: A fast algorithm to unmix hyperspectral data," *IEEE Trans. Geosci. Remote Sens.*, vol. 43, no. 4, pp. 898–910, Apr. 2005.
[30] L. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Phys. D Nonlinear Phenom.*, vol. 60, no. 1–4, pp. 259–268, 1992.
[31] A. Chambolle, "An algorithm for total variation minimization and applications," *J. Math. Imag. Vis.*, vol. 20, pp. 89–97, 2004.
[32] M. Afonso, J. Bioucas-Dias, and M. Figueiredo, "An augmented Lagrangian approach to the constrained optimization formulation of imaging inverse problems," *IEEE Trans. Image Process.*, vol. 20, no. 3, pp. 681–695, Mar. 2011.
[33] J. Eckstein and D. Bertsekas, "On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators," *Math. Program.*, vol. 5, pp. 293–318, 1992.
[34] O. Axelsson, *Iterative Solution Methods*. Cambridge, U.K.: Cambridge Univ. Press, 1996.
[35] G. S. Miller, "The definition and rendering of terrain maps," in *Proc. ACM SIGGRAPH Comput. Graph.*, vol. 20, no. 4. ACM, New York, NY, USA, Aug. 1986, pp. 39–48, doi: 10.1145/15886.15890.

Sergio Bernabé received the Computer Engineering degree and the M.Sc. degree in computer engineering from the University of Extremadura, Cáceres, Spain, in 2010, and the joint Ph.D. degree between the University of Iceland, Reykjavík, Iceland, and the University of Extremadura, Cáceres, Spain, in 2014.

He has been a Visiting Researcher at the Institute for Applied Microelectronics, University of Las Palmas de Gran Canaria, Las Palmas, Spain, and also at the Computer Vision Laboratory (LVC), Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil. He is a Postdoctoral Researcher (funded by FCT) with the Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal. His research interests include the development and efficient processing of parallel techniques for different types of high-performance computing architectures.

Dr. Bernabé currently serves as an Active Reviewer of international conferences and international journals, including the IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATION AND REMOTE SENSING, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, and the IEEE GEOSCIENCE AND REMOTE SENSING LETTERS. He was the recipient of the 2013 Best Paper Award of the JSTARS journal and the Best Ph.D. Dissertation award at the University of Extremadura in 2015.

Gabriel Martín received the Computer Engineering, the M.Sc., and the Ph.D. degrees from the University of Extremadura, Plasencia, Spain, in 2008, 2010, and 2013, respectively.

He was a Predoctoral Research Associate (funded by the Spanish Ministry of Science and Innovation) with the Hyperspectral Computing Laboratory, University of Extremadura, and is now a Postdoctoral Researcher (funded by FCT) with the Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal. His research interests include the development of new techniques for unmixing remotely sensed hyperspectral data sets, as well as the efficient processing and interpretation of these data in different types of high-performance computing architectures.

Dr. Martín has served as a Reviewer for the IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING and the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING.

José M. P. Nascimento (S'03–M'06) received the B.S. degree in electronics and telecommunications engineering and E.E. degree in electrical engineering from the Instituto Superior de Engenharia de Lisboa, Institute Politechnic of Lisbon, Lisbon, Portugal, in 1993 and 1995, respectively, and the M.Sc. and Ph.D. degrees in electrical and computer engineering from the Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal, in 2000 and 2006, respectively.

Currently, he is a Professor with the Department of Electronics, Telecommunications, and Computer Engineering, Instituto Superior de Engenharia de Lisboa, Lisbon, Portugal. He is also a Researcher with the Instituto de Telecomunicações, Lisbon, Portugal. Recently, he is a Principal Researcher of two projects in the field of high-performance computing for hyperspectral imagery. His research interests include remote sensing, image processing, and high-performance computing.

Dr. Nascimento is currently serving as a Reviewer of several international journals and he has also been a member of program/technical committees of several international conferences.

José M. Bioucas-Dias (S'87–M'95) received the E.E., M.Sc., Ph.D., and "Agregado" degrees in electrical and computer engineering from the Instituto Superior Técnico (IST), Engineering School of the Technical University of Lisbon (TULisbon), Lisbon, Portugal, in 1985, 1991, 1995, and 2007, respectively.

Since 1995, he has been with the Department of Electrical and Computer Engineering, IST, where he was an Assistant Professor from 1995 to 2007 and has been an Associate Professor since 2007. Since 1993, he has been also a Senior Researcher with the Pattern and Image Analysis Group, Instituto de Telecomunicações, Lisbon, Portugal, which is a private nonprofit research institution. His research interests include inverse problems, signal and image processing, pattern recognition, optimization, and remote sensing.

Dr. Bioucas-Dias was an Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS from 1997 to 2000, and is an Associate Editor for the IEEE TRANSACTIONS ON IMAGE PROCESSING and the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING. He was a Guest Editor of the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING for the Special Issue on Spectral Unmixing of Remotely Sensed Data and of the IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING for the Issue on Hyperspectral Image and Signal Processing, and he is a Guest Editor of the IEEE SIGNAL PROCESSING MAGAZINE for the Special Issue on Signal and Image Processing in Hyperspectral Remote Sensing. He was the General Co-Chair of the third IEEE GRSS Workshop on Hyperspectral Image and Signal Processing, Evolution in Remote sensing (WHISPERS'2011) and has been a member of program/technical committees of several international conferences.

Antonio Plaza (SM'05) was born in Caceres, Spain, in 1975.

He is an Associate Professor (with accreditation for Full Professor) with the Department of Technology of Computers and Communications, University of Extremadura, Plasencia, Spain, where he is the Head of the Hyperspectral Computing Laboratory (HyperComp). He has been the Advisor of 12 Ph.D. dissertations and more than 30 M.Sc. dissertations. He was the Coordinator of the Hyperspectral Imaging Network and a European project with total funding of 2.8 million Euros. He has authored more than 400 publications, including 126 JCR journal papers (78 in IEEE journals), 20 book chapters, and over 240 peer-reviewed conference proceeding papers (94 in IEEE conferences). He has edited the book *High-Performance Computing in Remote Sensing* (CRC Press/Taylor and Francis) (the first book on this topic in the published literature) and guest edited eight special issues on hyperspectral remote sensing for different journals. His research interests include remotely sensed hyperspectral image analysis and efficient implementations of large-scale scientific problems on high-performance computing architectures.

Dr. Plaza served as an Associate Editor for the IEEE GEOSCIENCE AND REMOTE SENSING LETTERS from 2007 to 2012. He is also an Associate Editor for the IEEE ACCESS, and was a member of the Editorial Board of the IEEE GEOSCIENCE AND REMOTE SENSING NEWSLETTER from 2011 to 2012 and the IEEE GEOSCIENCE AND REMOTE SENSING MAGAZINE in 2013. He was also a member of the steering committee of the IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING (JSTARS). Currently, he is an Associate Editor for the *Journal of Real-Time Image Processing*. He served as the Director of Education Activities for the IEEE Geoscience and Remote Sensing Society (GRSS), from 2011 to 2012, and is currently serving as a President of the Spanish Chapter of IEEE GRSS (since November 2012). He has served as a proposal evaluator for the European Commission (Marie Curie Actions, Engineering Panel), the European Space Agency, the Belgium Science Policy, the Israel Science Foundation, and the Spanish Ministry of Science and Innovation. He has reviewed more than 500 manuscripts for over 50 different journals. He is also currently serving as the Editor-in-Chief of the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING JOURNAL. He was the recipient of the recognition of Best Reviewers of the IEEE GEOSCIENCE AND REMOTE SENSING LETTERS in 2009 and the recognition of Best Reviewers of the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING in 2010, the 2013 Best Paper Award of the JSTARS journal, and a recipient of the most highly cited paper from 2005 to 2010 in the *Journal of Parallel and Distributed Computing*. He is the coauthor of the 2011 Best Student Paper at the IEEE International Conference on Space Technology, and a recipient of the 2008 Best Paper award at the IEEE Symposium on Signal Processing and Information Technology. He was the recipient of the Best Ph.D. Dissertation award at the University of Extremadura in 2002. He has participated in the Tenure Track Selection Committee of different universities in Italy, Spain, and Australia.

Vítor Silva received the Licenciado and Ph.D. degrees in electrical engineering from the University of Coimbra, Coimbra, Portugal, in 1984 and 1996, respectively.

Currently, he is an Auxiliary Professor with the Department of Electrical and Computer Engineering, University of Coimbra. He has contributed to more than 120 papers in leading international journals and conferences. His research interests include signal processing for communications, parallel computing using heterogeneous and reconfigurable architectures, coding theory and image and video compression are mainly carried out at the Instituto de Telecomunicações, Coimbra, Portugal, where is the Head of the Multimedia Signal Processing Research Group.