A Fast Parallel Hyperspectral Coded Aperture Algorithm for Compressive Sensing Using OpenCL

Sergio Bernabé[1], Gabriel Martín[2], José M. P. Nascimento[2], José M. Bioucas-Dias[2], Antonio Plaza[3], Guillermo Botella[1] and Manuel Prieto-Matías[1]

***Abstract*—In this paper, we develop a fast implementation of an hyperspectral coded aperture (HYCA) algorithm on different platforms using OpenCL, an open standard for parallel programing on heterogeneous systems, which includes a wide variety of devices, from dense multicore systems from major manufactures such as Intel or ARM to new accelerators such as graphics processing units (GPUs), field programmable gate arrays (FPGAs), the Intel Xeon Phi and other custom devices. Our proposed implementation of HYCA significantly reduces its computational cost. Our experiments have been conducted using simulated data and reveal considerable acceleration factors. This kind of implementations with the same descriptive language on different architectures are very important in order to really calibrate the possibility of using heterogeneous platforms for efficient hyperspectral imaging processing in real remote sensing missions.**

***Keywords*—Hyperspectral imaging, compressive sensing (CS), coded aperture, high performance computing (HPC), OpenCL.**

I. Introduction

HYPERSPECTRAL imaging is concerned with the extraction of information from objects or scenes lying on the Earth surface, using hundreds of (narrow) spectral bands typically covering the visible and near infra-red domains [1]. In hyperspectral imaging, also termed imaging spectroscopy [2], the sensor acquires a spectral vector with hundreds or thousands of elements from every pixel in a given scene. The result is the so-called hyperspectral image or hyperspectral data *cube*. It should be noted that hyperspectral images are spectrally smooth and spatially piece-wise smooth; this means that the values in neighboring locations and wavelengths are often highly correlated. 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].

In addition to extremely large dimensionality, another 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. 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 [6], [7]. However, 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.

In contrast, 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 [8], where the acquisition of the data in an already compressed form is involved. The hyperspectral coded aperture (HYCA) method, which combine the ideas of spectral unmixing and compressive sensing has been recently proposed in [9]. It takes advantage of two main properties of hyperspectral data, namely 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. However, HYCA method is computational expensive and time consuming, a fact that compromises its use in applications under real-time constraints. To overcome this problem, in [10] a parallel version implemented on GPUs is presented.

In this paper we have augmented the previous implementation on different platforms using OpenCL, an open and royalty-free standard based on C99 for parallel programing on heterogeneous systems. Our experiments have been conducted using simulated data and reveal considerable acceleration factors. The present implementation on different architectures but using the same programming paradigm open the door to the viability study of heterogeneous platforms for efficient hyperspectral imaging processing in real remote sensing missions.

The remainder of this paper is organized as follows. Section II describes the original HYCA method. Section III describes the proposed parallel implementation. Section IV presents an experimental evaluation of the proposed implementation in terms of both accuracy and parallel performance using

[1]Sergio Bernabé, Guillermo Botella and Manuel Prieto-Matías are with the Department of Computer Architecture and Automation, Complutense University, Madrid, 28040 Spain. E-mail: sebernab@ucm.es.

[2]Gabriel Martín, José M. P. Nascimento and José M. Bioucas-Dias are with the Instituto de Telecomunicações, Instituto Superior Técnico, Universidade de Lisboa, 1049-001 Lisbon, Portugal.

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

978-1-4799-8569-2/15/$31.00 ©2015 IEEE

simulated data on heterogeneous platforms. Finally, Section VI presents as few concluding remarks and pointers to future work.

II. Method description and its fast optimization

The original HYCA method for CS was developed in [9]. 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 the fact that the hyperspectral data can be explained 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. The algorithm can be briefly summarized as follows.

Let $\mathbf{x} \in \mathbb{R}^{n_b \times n_p}$ represent, in vector format, a hyperspectral image with n_b spectral bands and $n_p := n_r \times n_c$ pixels where n_r and n_c denote, respectively, the number of rows and columns of the hyperspectral image in the spatial domain. The ordering of $\mathbf{x}$ correspond to all image pixels for each spectral band. In order to perform the compression of the original signal $\mathbf{x}$, and as in [9], 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 $\mathbf{y} = \mathbf{A}\mathbf{x}$, 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 a) the computational management of the sampling process and b) the spatial correlation length of hyperspectral images (see [9] 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 $j = 1, \ldots, m$.

Let us now define the linear operator $\mathbf{x} = (\mathbf{I} \otimes \mathbf{E})\mathbf{z}$, where the matrix $\mathbf{E}$ represents the basis of the subspace where the data lives [11], $\mathbf{I}$ is the identity matrix and the vector $\mathbf{z}$ contains the coefficients. In this work, the $\mathbf{E}$ matrix contains the p endmembers of the data set by columns obtained in a very fast way through the vertex component analysis (VCA) algorithm [12], thus $\mathbf{z}$ contains the fractional abundances associated to each pixel.

Let us now assume that $\mathbf{K} = \mathbf{H}(\mathbf{I} \otimes \mathbf{E})$. If matrices $\mathbf{E}$ and $\mathbf{H}$ are available, one can formulate the estimation of $\mathbf{z}$ with from q-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:

$$\min_{\mathbf{z} \geq \mathbf{0}} (1/2)\|\mathbf{y} - \mathbf{K}\mathbf{z}\|^2 + \lambda_{TV}\mathrm{TV}(\mathbf{z}). \tag{1}$$

where $\mathrm{TV}(\mathbf{z})$ stands for the sum of non-isotropic total variations (TV) [13], [14] associated to $\mathbf{z}$, one per image of abundance. Defined as:

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

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

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

with $\vartheta := [\vartheta_h^T, \vartheta_v^T]$, ϑ_h standing for horizontal differences and ϑ_v standing for vertical differences. The TV regularizer promotes piecewise abundance images $\mathbf{z}$. Therefore, the minimization (1) 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 paremeter $\lambda_{TV} > 0$. On the other hand, $\mathrm{soft}(\cdot, \tau)$ denotes the application of the soft-threshold function $\mathbf{b} \mapsto \mathbf{b}\frac{\max\{\|\mathbf{b}\|_2 - \tau, 0\}}{\max\{\|\mathbf{b}\|_2 - \tau, 0\} + \tau}$.

To solve the convex optimization problem in Eq. (1), a set of new variables per term of the objective function were used and the ADMM methodology [15] has been adopted to decompose very hard problems into a cyclic sequence of simpler problems. With this in mind, an equivalent way of writing the optimization problem in Eq. (1) is

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

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 (2), we can write the following equivalent formulation:

$$\begin{aligned} \min_{\mathbf{z},\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3,\mathbf{v}_4} \quad & \frac{1}{2}||\mathbf{y} - \mathbf{K}\mathbf{v}_1||^2 + \iota_{R+}(\mathbf{v}_2) + \lambda_{TV}\, \phi(\mathbf{D}\mathbf{z}) \\ \text{subject to} \quad & \mathbf{v}_1 = \mathbf{z} \\ & \mathbf{v}_2 = \mathbf{z} \\ & (\mathbf{v}_3, \mathbf{v}_4) = \mathbf{D}\mathbf{z}, \end{aligned} \tag{3}$$

Algorithm 1 shows the pseudo-code of the HYCA algorithm to solve the problem in (3) and how to reconstruct the data using the linear mixture model.

Algorithmically, the above pseudocode can be improved. The step 4 of Algorithm 1 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 2D cyclic convolutions, then the computation of $\mathbf{b}$ and the solution of the linear system of equations may be implemented efficiently in the frecuency 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 in $O(pn_p)$, it may be advantageous to solve the system $\mathbf{M}\mathbf{z} = \mathbf{b}$ with a first order stationary iterative procedure [16], which to run the following iterative procedure:

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. $\mathbf{A} \leftarrow (\mathbf{v}_1^{(k)} + \mathbf{d}_1^{(k)} + \mathbf{v}_2^{(k)} + \mathbf{d}_2^{(k)} + \mathbf{D}_h^T(\mathbf{v}_3^{(k)} + \mathbf{d}_3^{(k)})$
4. $\mathbf{z}^{(k+1)} \leftarrow (\mathbf{D}^T\mathbf{D} + 2\mathbf{I})^{-1}\left(\mathbf{A} + \mathbf{D}_v^T(\mathbf{v}_4^{(k)} + \mathbf{d}_4^{(k)})\right)$
5. $\mathbf{v}_1^{(k+1)} \leftarrow (\mathbf{K}^T\mathbf{K} + \mu\mathbf{I})^{-1}\left(\mathbf{K}^T\mathbf{y} + \mu(\mathbf{z}^{(k+1)} - \mathbf{d}_1^{(k)})\right)$
6. $\mathbf{v}_2^{(k+1)} \leftarrow \max\left(0, \mathbf{z}^{(k+1)} - \mathbf{d}_2^{(k)}\right)$
7. $\mathbf{v}_3^{(k+1)} \leftarrow \text{soft}\left(\mathbf{D}_h(\mathbf{z}^{(k+1)}) - \mathbf{d}_3^{(k)}, \lambda_{TV}/\mu\right)$
8. $\mathbf{v}_4^{(k+1)} \leftarrow \text{soft}\left(\mathbf{D}_v(\mathbf{z}^{(k+1)}) - \mathbf{d}_4^{(k)}, \lambda_{TV}/\mu\right)$
9. Update Lagrange multipliers:
$\mathbf{d}_1^{(k+1)} \leftarrow \mathbf{d}_1^{(k)} - \mathbf{z}^{(k+1)} + \mathbf{v}_1^{(k+1)}$
$\mathbf{d}_2^{(k+1)} \leftarrow \mathbf{d}_2^{(k)} - \mathbf{z}^{(k+1)} + \mathbf{v}_2^{(k+1)}$
$\mathbf{d}_3^{(k+1)} \leftarrow \mathbf{d}_3^{(k)} - \mathbf{D}_h\mathbf{z}^{(k+1)} + \mathbf{v}_3^{(k+1)}$
$\mathbf{d}_4^{(k+1)} \leftarrow \mathbf{d}_4^{(k)} - \mathbf{D}_v\mathbf{z}^{(k+1)} + \mathbf{v}_4^{(k+1)}$
10. Update iteration: $k \leftarrow k + 1$
11. until k = MAX_ITERATIONS
12. Reconstruction $\hat{\mathbf{x}} = (\mathbf{I} \otimes \mathbf{E})\mathbf{z}^k$

$$\textbf{for } t = 0, 1, \ldots \tag{4}$$

$$\mathbf{r}_t = \mathbf{M}\mathbf{z}_t - \mathbf{b} \tag{5}$$

$$\mathbf{z}_{t+1} = \mathbf{z}_t - \beta\mathbf{r}_t. \tag{6}$$

Let $0 < \lambda_{min} < \lambda_{max}$ denote, 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 $0 < \beta < 2/\lambda_{max}$ [16]. In addition, the optimal convergence factor is given by

$$\rho_{opt} = \frac{1 - \lambda_{min}/\lambda_{max}}{1 + \lambda_{min}\lambda_{max}} \tag{7}$$

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 $-1/\log(2/3) = 5.67$. In practice, is not necessary to solve exactly the linear system of equations in each ADMM iteration as far as the errors are summable [15]. In ADMM iteration, we initialize (4) with $\mathbf{z}^{(k)}$ and run only a few iterations.

The fast optimization of HYCA is termed HYCA-FAST hereinafter. The pseudocode with the main modification in line 4 of Algorithm 1 is shown in Algorithm 2. 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.

III. OPENCL FRAMEWORK AND IMPLEMENTATION

In this section we describe our mapping of the P-HYCA-FAST algorithm on different heterogeneous platforms. One of our goals is to use a common code written in OpenCL for the different platforms. OpenCL is a framework for parallel implementation that allows the execution of parallel programs on heterogeneous platforms. It is currently supported by several hardware devices, such as CPUs, GPUs, DSPs, FPGAs and other processors. OpenCL is based on the host-device model,

Algorithm 2 Pseudocode of FAST optimization.

...
3. Fast optimization:
3.1. set $\beta = 1/6$ **(optimum value)**
3.2. $\mathbf{A} \leftarrow (\mathbf{v}_1^{(k)} + \mathbf{d}_1^{(k)} + \mathbf{v}_2^{(k)} + \mathbf{d}_2^{(k)} + \mathbf{D}_h^T(\mathbf{v}_3^{(k)} + \mathbf{d}_3^{(k)})$
3.3. $\mathbf{g}^{(k)} = \left(\mathbf{A} + \mathbf{D}_v^T(\mathbf{v}_4^{(k)} + \mathbf{d}_4^{(k)})\right)$
3.4. repeat:
3.4.1. $\mathbf{r}^{(f)} = 2 * \mathbf{z}^{(f)} + \mathbf{D}_h^T(\mathbf{D}_h(\mathbf{z}^{(f)})) + \mathbf{D}_v^T(\mathbf{D}_v(\mathbf{z}^{(f)})) - \mathbf{g}^{(k)}$
3.4.2. $\mathbf{z}^{(f+1)} = \mathbf{z}^{(f)} - \beta * \mathbf{r}^{(f)}$
3.5. until f = DESIRED_ITERATIONS
...

where the host is in charge of device memory management, data transfer from/to device and kernel code invocation.

The kernel is a piece of code which expresses the parallelism of a program. The OpenCL programming model divides a program workload into *work-groups* and *work-items*. *Work-items* are grouped into a *work-group*, which is executed independently with respect to other *work-groups*. Data-level parallelism is regularly exploited in an SIMD way, in which several *work-items* are grouped according to the lane width capabilities of the target device.

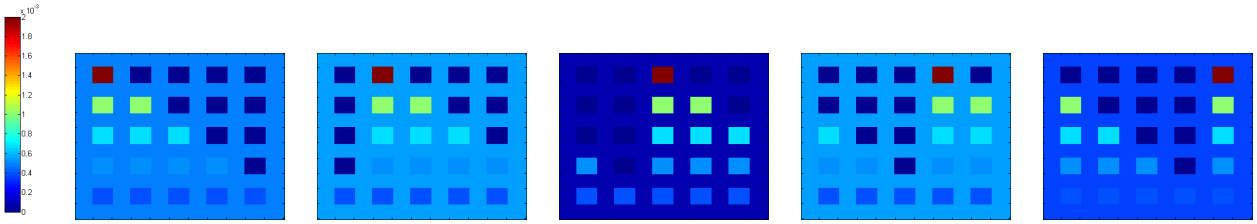
The OpenCL memory model distinguishes different memory regions that are characterized by the access type, performance and scope. Global memory is read-write accessible by all *work-items* across all *work-groups*, and it usually corresponds to the DRAM memory device, which carries a high latency memory access. Local memory is a shared read-write memory accessible from all work-items of a single *work-group*, and it habitually involves a low latency memory access. Constant memory is a read-only memory that is visible to all *work-items* across all *work-groups*, and private memory, as the name suggests, is only accessible by a single *work-item*.

As OpenCL is a cross platform standard for parallel programming on heterogeneous platforms, the developer can thus focus on algorithmic specifications, avoiding implementation details.

Before performing any processing on the platform, the compressed hyperspectral image is loaded band by band from the host to the device global memory. With this data layout, the access to consecutive pixels in the same wavelength performed by adyacent work-items, can be coalesced into fewer memory transactions.

The next step pre-computes 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 2-11. Herein, the term $(\mathbf{K}^T\mathbf{K} + \mu\mathbf{I})^{-1}$ in line 5 of Algorithm 1 is computed in the CPU using the LAPACK[1] package due to the fact that the size of these matrices is small and it is not worth to perform this computation in the parallel platform. 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. In this kernel the matrix $\mathbf{K}$ is stored in local memory to optimize the memory access to the elements of this matrix. It is important to emphasize that we have declared private memory dynamically in all kernel launch

[1] http://www.netlib.org/lapack/



(a) Endmember #1 (b) Endmember #2 (c) Endmember #3 (d) Endmember #4 (e) Endmember #5

Fig. 1. Ground-truch abundance maps of endmembers in the simulated data represented in gray scale.

configurations. In this case the private memory allocation (size per work-group) must be specified (in bytes) using an additional argument in the kernel.

The optimization of $\mathbf{z}$ in line 4 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 work-item computes one element. Later the optimization with respect to $\mathbf{z}$ is performed using the kernel called `Compute_z_Fast` to compute lines 3.3.1 and 3.3.2 of Algorithm 2. The kernel uses the same index space configuration as the previous ones (one work-item per pixel). This kernel is called f times and performs the update of both $\mathbf{r}$ and $\mathbf{z}$ variables. This algorithmic optimization avoids the use of libraries to perform Fourier transforms.

The optimization of $\mathbf{v}_1$ in line 5 of Algorithm 1 is carried out with a kernel called `Optimize_v1`. This kernel uses the same index space configuration as the previous ones. This kernel also makes use of the local memory to optimize the memory access to the matrix $(\mathbf{K}^T\mathbf{K} + \mu\mathbf{I})^{-1}$, which was pre-computed before. The resulting $\mathbf{v}_1$ is stored in the global memory.

Line 6 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 work-items as the number of elements of the vector (n_p), with the same index space 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 7 and 8 in Algorithm 1. This kernel uses the same configuration as the previous kernels with as many work-items as the image pixels. In this kernel each work-item computes the horizontal and vertical differences of one element and performs the `soft` function for the corresponding element.

Finally, the update Lagrange multipliers 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.

IV. Experimental Results

The experiments have been performed using simulated data generated from spectral signatures randomly selected from the United States Geological Survey (USGS)[2]. They consist of a set of 5×5 squares of 10×10 pixels each one, yielding a total size of 110×110 pixels (10.8 MB) and 224 spectral bands. 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. 1 displays the ground-truth abundance maps used for generating the simulated imagery.

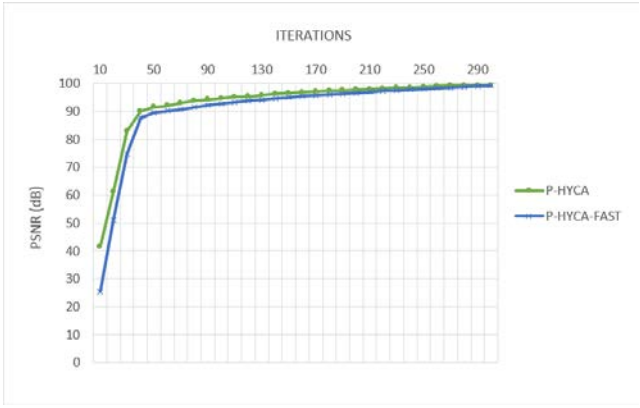


Fig. 2. PSNR between the original and the reconstructed images for P-HYCA and P-HYCA-FAST methods for values of $q = 5$ and $ws = 4$.

A. Analysis of Accuracy

In order to evaluate the accuracy of the proposed implementations in terms of reconstruction error, the peak signal-to-noise ratio (PSNR) has been adopted as performance indicator. The equation is given by

$$\text{PSNR} = 10 \log_{10} \frac{\max(\mathbf{x})^2 \times n_p \times n_b}{||\widehat{\mathbf{x}} - \mathbf{x}||_F^2}. \quad (8)$$

Fig. 2 shows the PSNR achieved for the proposed implementation as a function of the number of iterations. The parameter values were defined empirically. In this experiment one can see that P-HYCA and their fast version provide very similar values when the number of iterations is high. On the other hand, P-HYCA-FAST presents highest signal-to-noise ratio in the simulated data demostrating their good performance.

[2]http://speclab.cr.usgs.gov

TABLE I. OPENCL FEATURE

Device Type	Model	Compute Units	Clock Freq.	Global Mem. Size	Local Mem. Size	Max *work-items*
CPU	2×Xeon E5-2695	56	2.3 GHz	64GB	32KB	8192×8192×8192
GPU	NVIDIA-K20c	13	705 MHz	4.8GB	48KB	1024×1024×64
Accelerator	Intel Xeon Phi	228	1.1 GHz	6GB	32KB	8192×8192×8192

B. Analysis of Parallel Performance

To carry out the tests on OpenCL, we have used a multicore heterogeneous server equipped with two Intel Xeon E5-2695 processors with 14 cores each, running at 2.30GHz, with 64 GB of DDR3 RAM memory. The Operating System used is GNU-Linux (CentOS release 6.6). Additionally, we also used both GPU and accelerator based on NVIDIA-K20c and Intel Xeon Phi 3120P. The Table I summarized the main features of the system used for the comparison.

TABLE II. PROCESSING TIMES (IN SECONDS) AND SPEEDUPS ACHIEVED FOR THE P-HYCA-FAST IN THE DIFFERENT PLATFORMS AND TESTED WITH SIMULATED DATA.

	SYNTHETIC data set		
	Dual Xeon	**GPU-K20c**	**Xeon Phi**
HOST → $DeviceGlobalMem.$	0.0002	0.0001	0.0016
$(\mathbf{K^T K} + \mu \mathbf{I})^{-1}$	0,0002	0.0002	0.0003
HOST → $DeviceGlobalMem.$	≈0.0000	≈0.0000	0.0017
$(\mathbf{K}^T \mathbf{y})$	0.0013	0.0005	0.0091
Compute $\mathbf{z}$ (lines 3 - 4)	1.3368	0.0206	1.0330
Compute $\mathbf{v}_1$ (line 5)	0.2674	0.0033	0.1977
Compute $\mathbf{v}_2$ (line 6)	0.2680	0.0044	0.1958
Compute $\mathbf{v}_3$, $\mathbf{v}_4$ (lines 7 - 8)	0.2652	0.0031	0.1953
Compute $\mathbf{d}$ (line 9)	0.5336	0.0088	0.3937
Reconstruction (line 12)	0.0151	0.0268	0.0135
HOST ← $DeviceGlobalMem.$	≈0.0000	≈0.0000	≈0.0000
Total Time	2.6890	0.0678	2.0416
Speedup (Xeon Time / GPU‖Xeon Phi Time)	–	39.66x	1.32x

Before describing the performance of the parallel implementation, it is important to emphasize that it provides exactly the same results than the serial counterpart. Hence, the only difference between the serial and parallel version is the time they need to complete their calculations. For each experiment, ten runs were performed and the mean values were reported (these times were always very similar, with differences of the order of a few milliseconds). Table II summarizes the obtained results after processing simulated data on the considered platforms during 175 iterations. From this point, the accuracy between P-HYCA and P-HYCA-FAST are very similar. The implementation on the GPU platform outperforms the Xeon and Xeon Phi counterparts. The speedup factor over the baseline Xeon implementation is 39×.

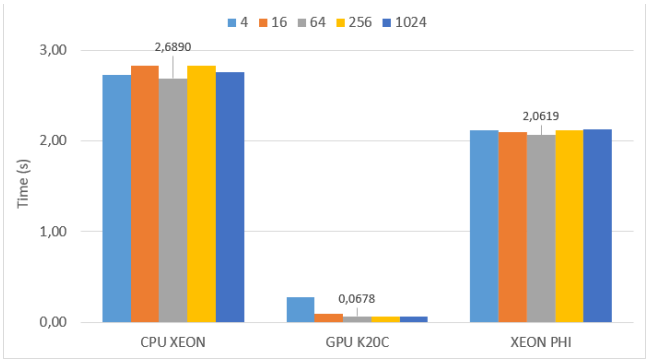


Fig. 3. Execution times achieved on CPU Xeon, NVIDIA's GPU and Xeon Phi using different work-group sizes.

Fig. 3 shows the impact of the work-group size on performance. As can be seen, this parameter is important on the GPU device, but on the other platforms, the effects of this parameter are negligible.

V. ACKNOWLEDGEMENT

This work has been supported by the Spanish Ministry of Economy and Competitiveness (MINECO) through the research contract TIN 2012-32180 and the Formación Posdoctoral programme (FPDI-2013-16280), and by the Portuguese Science and Technology Foundation under Projects UID/EEA/50008/2013 and SFRH/BPD/94160/2013.

VI. CONCLUSION

In this paper, an OpenCL implementation of a hyperspectral coded aperture algorithm for compressive is presented. The use of OpenCL introduces a level of freedom in the adoption of different high-performance computing solutions. The implementation exploits a fast algorithmic optimization and several optimizations to better use the underlying memory. The results obtained on the conducted experiments reveal considerable acceleration factors, which can satisfy the real-time requirements. Further experimentation with additional real hyperspectral scenes and high performance computing architectures (such as field programmable gate arrays or digital signal processors) are desirable in future developments 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 Geoscience and Remote Sensing Magazine*, vol. 1, pp. 6–36, 2013.

[2] A. F. H. Goetz, G. Vane, J. E. Solomon, and B. N. Rock, "Imaging spectrometry for Earth remote sensing," *Science*, vol. 228, pp. 1147–1153, 1985.

[3] A. Plaza and C.-I. Chang, *High Performance Computing in Remote Sensing*. Taylor & Francis: Boca Raton, FL, 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 Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 5, no. 2, pp. 354–379, 2012.

[5] Y. Altmann, N. Dobigeon, and J.-Y. Tourneret, "Unsupervised post-nonlinear unmixing of hyperspectral images using a hamiltonian monte carlo algorithm," *IEEE Transactions on Image Processing*, vol. 23, no. 6, pp. 2663–2675, 2014.

[6] 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 Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 6, no. 3, pp. 1386–1398, 2013.

[7] S. Bernabe, S. Lopez, A. Plaza, and R. Sarmiento, "GPU Implementation of an Automatic Target Detection and Classification Algorithm for Hyperspectral Image Analysis," *IEEE Geoscience and Remote Sensing Letters*, vol. 10, no. 2, pp. 221–225, 2013.

[8] P. V. M. Golbabaee, S. Arberet, "Compressive source separation: Theory and methods for hyperspectral imaging," *arXiv preprint arXiv:1208.4505*, 2012.

[9] G. Martin, J. M. Bioucas-Dias, and A. Plaza, "Hyperspectral coded aperture (HYCA): A new technique for hyperspectral copressive sensing," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 5, pp. 2819–2831, 2015.

[10] S. Bernabe, G. Martin, J. M. P. Nascimento, J. M. Bioucas-Dias, A. Plaza, and V. Silva, "GPU Implementation of a Hyperspectral Coded Aperture Algorithm for Compressive Sensing," *Proceedings of the IEEE Geoscience and Remote Sensing Symposium (IGARSS'15), Milan, Italy*, 2015.

[11] J. M. Bioucas-Dias and J. M. P. Nascimento, "Hyperspectral subspace identification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 46, no. 8, pp. 2435–2445, 2008.

[12] J. M. P. Nascimento and J. M. Bioucas-Dias, "Vertex Component Analysis: A Fast Algorithm to Unmix Hyperspectral Data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 43, no. 4, pp. 898–910, 2005.

[13] L. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D: Nonlinear Phenomena*, vol. 60, no. 1-4, pp. 259–268, 1992.

[14] A. Chambolle, "An algorithm for total variation minimization and applications," *Journal of Mathematical Imaging and Vision*, vol. 20, pp. 89–97, 2004.

[15] J. Eckstein and D. Bertsekas, "On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators," *Mathematical Programming*, vol. 5, pp. 293–318, 1992.

[16] O. Axelsson, *Iterative Solution Methods*. New York: Cambridge University Press, 1996.