GPU IMPLEMENTATION OF A HYPERSPECTRAL CODED APERTURE ALGORITHM FOR COMPRESSIVE SENSING

Sergio Bernabé[a], Gabriel Martín[a], José M. P. Nascimento[a,b], José M. Bioucas-Dias[a,c], Antonio Plaza[d] and Vítor Silva[a,e]

[a]Instituto de Telecomunicações, Lisbon, Portugal
[b]Instituto Superior de Engenharia de Lisboa, Lisbon, Portugal
[c]Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal
[d]Hyperspectral Computing Laboratory, University of Extremadura, Cáceres, Spain
[e]Instituto de Telecomunicações, DEEC, University of Coimbra, Coimbra, Portugal

ABSTRACT

This paper presents a new parallel implementation of a previously hyperspectral coded aperture (HYCA) algorithm for compressive sensing on graphics processing units (GPUs). HYCA method combines the ideas of spectral unmixing and compressive sensing exploiting the high spatial correlation that can be observed in the data and the generally low number of endmembers needed in order to explain the data. The proposed implementation exploits the GPU architecture at low level, thus taking full advantage of the computational power of GPUs using shared memory and coalesced accesses to memory. The proposed algorithm is evaluated not only in terms of reconstruction error but also in terms of computational performance using two different GPU architectures by NVIDIA: GeForce GTX 590 and GeForce GTX TITAN. Experimental results using real data reveals signficant speedups up with regards to serial implementation.

Index Terms— Hyperspectral imaging, compressive sensing (CS), coded aperture, graphics processing units (GPUS).

1. INTRODUCTION

Hyperspectral spectrometers collect hundreds or even thousands of nearly contiguous spectral bands of 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 micrometers) of reflected light in an area of 2 to 12 kilometers wide and several kilometers long, using 224 spectral bands. The resulting multidimensional data cube typically comprises several GB per flight.

One of the main problems in the analysis of hyperspectral data is the presence of mixed pixels, as many of the pixels collected by imaging spectrometers such as AVIRIS are highly mixed in nature due to spatial resolution and other phenomena. In this case, several spectrally pure signatures (endmembers) are combined into the same (mixed) pixel. Spectral unmixing [2] involves the separation of a pixel spectrum into its endmember spectral, and the estimation of the abundance value for each endmember. The linear mixture model has been the most popular tool used to unmix remotely sensed hyperspectral data [3]. In recent years, several approaches have been proposed to solve the aforementioned problem using high performance computing systems: commodity clusters [4], field programmable gate arrays (FPGAs) [5], multicore processors [6] and commodity graphics processing units (GPUs) [7]. The last two alternatives offer highly relevant computational power at low cost, thus offering the opportunity to bridge the gap towards real-time analysis of remotely sensed hyperspectral data.

Due to the extremely large volumes of data collected by imaging spectrometers, hyperspectral data compression has received considerable interest in recent years [8, 9]. 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. In contrast, the application of compressive sensing (CS) to hyperspectral images is an active area of research, where the acquisition of the data in an already compressed form is involved. In [10], a hyperspectral coded aperture (HYCA) method is proposed, which combines the ideas of spectral unmixing and compressive sensing. It

This work was supported in part by the Instituto de Telecomunicações and in part by the Portuguese Science and Technology Foundation under Project UID/EEA/50008/2013, Project PTDC/EEI-PRO/1470/2012, and Project SFRH/BPD/94160/2013.

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. 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, a parallel implementation of HYCA [10] algorithm for compressive sensing on GPUs using the compute unified device architecture (CUDA) is proposed. The parallel hyperspectral coded aperture (P-HYCA) implementation exploits the GPU architecture at low level, using shared memory and coalesced accesses to memory. Experimental results conducted using real hyperspectral data set collected by the AVIRIS instrument and two different GPU architectures by NVIDIA: GeForce GTX 590 (GPU1) and GeForce GTX TITAN (GPU2), reveal that the use of GPUs can provide real-time compressive sensing performance.

The remainder of this paper is organized as follows. Section 2 briefly describes the original HYCA method. Section 3 describes the proposed GPU implementation. Section 4 presents an experimental evaluation of the proposed implementation in terms of both accuracy and parallel performance using real data on two GPU platforms. Finally, Section 5 outlines the conclusions of the paper with some remarks and pointers to future work.

2. DESCRIPTION OF THE METHOD

The original HYCA method for CS was developed in [10]. 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 $\mathbf{x} \in R^{n_b \times n_p}$ 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 [10], for each pixel $i \in \{1, \ldots, n_p\}$, a set of q inner products between $\mathbf{x}_i$ and samples of iid Gaussian random vectors is performed. The total number of measurements is therefore qn_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 $\mathbf{A}_i \in R^{q \times n_b}$ 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 [10] 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:

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

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 $\lambda_{TV} > 0$.

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 [13] 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.

Algorithm 1 Pseudocode of HYCA algorithm.

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

3. GPU IMPLEMENTATION

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). Fig. 1 shows the flowchart of the parallelized HYCA method by CUDA.

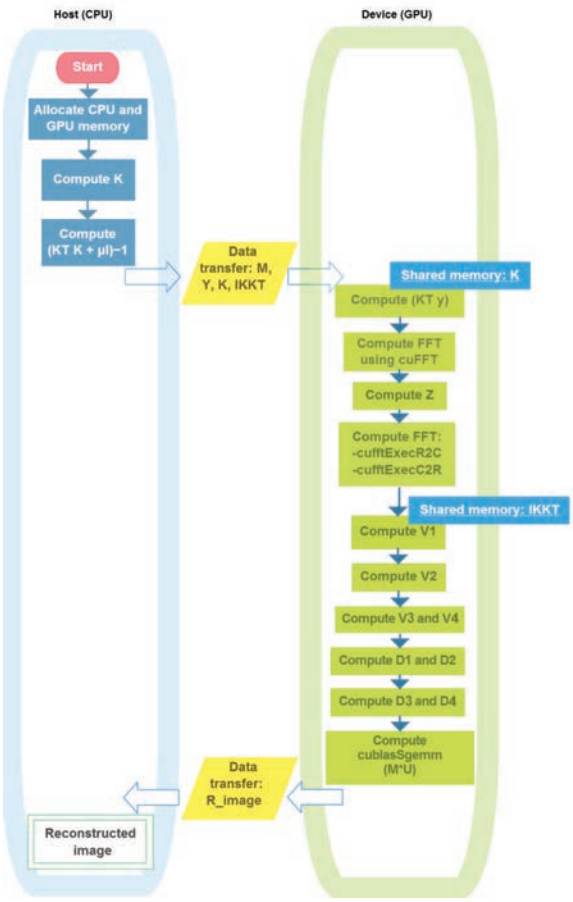

Fig. 1. The Flowchart of the parallelized HYCA method by CUDA.

4. EXPERIMENTAL RESULTS

The real hyperspectral image considered in experiments is the well-known AVIRIS Cuprite scene, available online in reflectance units after atmospheric correction[1]. 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 [11] and their signatures were extracted in a very fast way through VCA algorithm [12].

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) has been adopted as performance indicator. In this experiments, we bring the dimensionality of the original hyperspectral image from $n_b = 188$ to $q = 5$, thus achieving a compression ratio of $n_b/q = 37.60$. Fig. 2 shows that this method presents lowest error demostrating their good performance in real data sets. Note that the NMSE values obtained for the proposed parallel implementation present the same result as the serial version.

The proposed P-HYCA algorithm has been tested on two different GPUs: NVidia GeForce GTX 590[2] (GPU1) and GTX TITAN[3] (GPU2). The CPU platform is equipped with

[1] http://aviris.jpl.nasa.gov

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

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

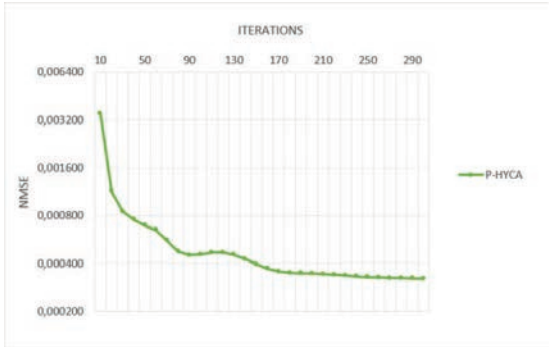

Fig. 2. NMSE achieved as a function of the number of iterations between the original and the reconstructed image for P-HYCA method and tested with the AVIRIS Cuprite scene.

a Intel i7-2600 CPU at 3.40 GHz with 4 physical cores and 16 GB of DDR3 RAM memory. It is important to emphasize that our GPU version provide exactly the same results as the serial version of the implemented algorithm. The gcc-4.8.2 (gnu compiler default) with optimization flags `-O3` (for the single-core version) and the 3.3.4 FFTW library[4] version have 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, while 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, ten 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 1 summarizes the obtained results after processing simulated data on the considered GPU platforms during 175 iterations. As shown in Table 1, the scene could be processed with significant speedup factor using the GPU2, up to 19 times, including both the processing time and the time spent on host/device memory transfers.

Table 1. Processing times (in seconds) and speedups achieved for the P-HYCA in two different platforms and tested with the AVIRIS Cuprite scene.

	AVIRIS Cuprite		
	CPU	**GPU1**	**GPU2**
RAM $\rightarrow GlobalMem.$	–	0.0007	0.0004
Compute P-HYCA	13.1022	0.8621	0.6484
RAM $\leftarrow GlobalMem.$	–	0.0110	0.0087
Total Time	13.1022	0.8738	0.6575
Speedup (CPU time / GPU time)	–	15.00x	19.93x

5. CONCLUSION AND REMARKS

In this paper, we presented a new parallel implementation of a hyperspectral coded aperture algorithm for compressive sensing on graphics processing units (GPUs). The method optimized exploits the GPU architecture at low level, thus taking full advantage of the computational power of GPUs using shared memory and coalesced accesses to 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.

[4]http://www.fftw.org/#documentation

6. 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] 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.

[3] N. Keshava and J. F. Mustard, "Spectral unmixing," *IEEE Signal Processing Magazine*, vol. 19, no. 1, pp. 44–57, 2002.

[4] A. Plaza, D. Valencia, J. Plaza, and P. Martinez, "Commodity cluster-based parallel processing of hyperspectral Imagery," *Journal of Parallel and Distributed Computing*, vol. 66, no. 3, pp. 345–358, 2006.

[5] A. Plaza and C.-I Chang, "Clusters versus FPGA for parallel processing of hyperspectral imagery," *International Journal of High Performance Computing Applications*, vol. 22, no. 4, pp. 366–385, 2008.

[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] G. Motta, F. Rizzo, and J. A. Storer, *Hyperspectral data compression*, Berlin: Springer, 2006.

[9] Bormin Huang, *Satellite data compression*, Berlin: Springer, 2011.

[10] 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.

[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] 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.