A Hybrid CPU-GPU Real-Time Hyperspectral Unmixing Chain

Emanuele Torti, *Member, IEEE*, Giovanni Danese *Member, IEEE*, Francesco Leporati, *Member, IEEE*, and Antonio Plaza, *Senior Member, IEEE*

***Abstract*—Hyperspectral images are used in different applications in Earth and space science, and many of these applications exhibit real- or near real-time constrains. A problem when analyzing hyperspectral images is that their spatial resolution is generally not enough to separate different spectrally pure constituents (*endmembers*); as a result, several of them can be found in the same pixel. Spectral unmixing is an important technique for hyperspectral data exploitation, aimed at finding the spectral signatures of the endmembers and their associated abundance fractions. The development of techniques able to provide unmixing results in real-time is a long desired goal in the hyperspectral imaging community. In this paper we describe a real-time hyperspectral unmixing chain based on three main steps: 1) estimation of the number of endmembers using the hyperspectral subspace identification with minimum error (HySime), 2) estimation of the spectral signatures of the endmembers using the vertex component analysis (VCA), and 3) unconstrained abundance estimation. We have developed new parallel implementations of the aforementioned algorithms and assembled them in a fully operative real-time unmixing chain using graphics processing units (GPUs), exploiting NVIDIA's compute unified device architecture (CUDA) and its basic linear algebra subroutines library (CuBLAS), as well as OpenMP and BLAS for multi-core parallelization. As a result, our real-time chain exploits both CPU (multi-core) and GPU paradigms in the optimization. Our experiments reveal that this hybrid GPU-CPU parallel implementation fully meets real-time constraints in hyperspectral imaging applications.**

***Index Terms*—Hyperspectral imaging, spectral unmixing, graphics processing units (GPUs), hyperspectral signal subspace identification with minimum error (HySime), vertex component analysis (VCA), abundance estimation.**

I. INTRODUCTION

Hyperspectral sensors are characterized by their high spectral resolution and for their capacity to collect massive data volumes. For instance, a typical "data cube", collected by the well-known NASA Jet Propulsion Laboratory's Airborne Visible Infrared Imaging Spectrometer (AVIRIS) [1], has a size of 614 lines with 512 samples and 224 spectral bands, with spectral resolution of 10 nm. This amounts to approximately 140 MB of data per cube. AVIRIS scanning rate is 12 Hz, but several instruments under development such as the Environmental Mapping and Analysis Program (EnMAP) [2] or Hyperion [3] are able to collect data at higher rates, such as 230 Hz for EnMAP and 220 Hz for Hyperion. With those scanning rates, a standard AVIRIS hyperspectral data cube should be processed in about 5 seconds in order to meet real-time processing performance.

An important aspect in hyperspectral image processing is that the pixel resolution is generally not fine enough to separate different pure constituent materials (called *endmembers* in hyperspectral jargon [4]). As a result, there is a need to develop real-time implementations of spectral unmixing techniques. Generally, two mixture models have been used for hyperspectral image analysis: *linear* [5] and *nonlinear* [6]. The first one is the most widely used, since it is simple to implement and unsupervised. In this model, each mixed pixel is interepreted as a linear combination of the endmember spectral signatures weighted by the correspondent percentage of pure material present in the pixel, i.e. the so called *abundance fraction*. Each pixel can be represented by a *N*-dimensional spectral vector, where *N* is the number of the acquired bands. At the spatial coordinates (i,j), the pixel vector can be represented as $s_{ij}=[s_{ij}(1),s_{ij}(2),...,s_{ij}(L)]\in\Re^L$, where $s_{ij}(k)$ is the spectral response of sensor channels $k=1,2,...,L$. Under these assumptions, the pixel vector at coordinates (i,j) is given by:

$$s_{ij}=Ma_{ij}+n_{ij}=\sum_{k=1}^{p}m_k a_{ij}^k+n_{ij} \quad , \tag{1}$$

where $M=[m_1,m_1,...,m_p]$ is a so-called *mixing matrix*, which contains the spectral signature of p endmembers, $a_{ij}=[a_{ij}^1,a_{ij}^2,...,a_{ij}^p]^T$ is the abundance vector made up of the abundance fractions of the p endmembers, and n_{ij} is an additive noise vector. The unmixing process consists of estimating the mixing matrix and the abundance fractions. For this operation, it is necessary to estimate first the number of endmembers in the data cube, which is a crucial task for correct data unmixing. Typically, the number of bands L is much greater than the number of endmembers p, and hyperspectral vectors lie in a lower-dimensional subspace [5].

In order to estimate p, many different approaches have been proposed in the literature, including projection methods for seeking the subspace through minimizing a cost function [7]; topological methods for finding the manifold where data set

Manuscript received January 21, 2015; revised June 2015 and September 2015; accepted September 18, 2015.

E. Torti, G. Danese, and F. Leporati are with the Dipartimento di Ingegneria Industriale e dell'Informazione, University of Pavia, 27100 Pavia, Italy (e-mail: emanuele.torti01@ateneopv.it, gianni.danese@unipv.it, francesco.leporati@unipv.it).

A. Plaza is with the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura, 10071 Caceres, Spain (e-mail: aplaza@unex.es).

lives [8]; and projection-based techniques which analyze the existing correlation between adjacent bands [9]. After the estimation of p, the endmember identification step can be performed. It is possible to distinguish between two main categories of endmember identification algorithms: with and without the pure pixel assumption [5]. Pure pixel-based algorithms are more widely used than the others, mainly because they have lower computational cost. Finally, for abundance estimation constrained, partially constrained and unconstrained algorithms can be used [4].

Performing the unmixing chain under real-time constraints is a highly desired goal in many applications, such as biological threat detection, monitoring of chemical contamination, wildfire tracking and so on. In recent years, some unmixing chain have been implemented using high performance computing architectures such as graphics processing units (GPUs) and multi-core processors [11, 12]. For instance, in [13] an unmixing chain based on the popular virtual dimensionality (VD) algorithm [14] for estimating the number of endmembers, orthogonal subspace projection with Gram-Schmidt orthogonalization (OSP-GS) [15] for estimating the endmember signatures, and unconstrained least squares (UCLS) [16] for abundance fraction estimation. However, it must be noted that more precise and reliable algorithms for hyperspectral image processing have been developed in recent years, such as hyperspectral subspace identification with minimum error (HySime) [10] for estimating the number of endmembers, or vertex component analysis (VCA) for estimating endmember signatures [17]. The HySime and VCA algorithms have been implemented in parallel [18, 19]. However, it is still possible to optimize those implementations and build a full spectral unmixing chain in real-time. In this paper, we develop a full spectral unmixing chain implemented in real-time using both GPUs and multi-core CPUs simultaneously (i.e., it is a hybrid CPU-GPU implementation), enhancing the parallel implementations of HySime and VCA and adding an implementation of abundance estimation by exploiting the computational power of both GPUs and multi-core processors, in a hybrid fashion. For the GPU part, we have used NVidia CUDA [20] and the CuBLAS[1] library, while for the multi-core part we have used OpenMP[2]. Moreover, we have also developed a multi-core version based on the popular BLAS functions provided by the GNU scientific library[3]. A main novelty of the proposed implementation with regard to other works in the literature is precisely its hybrid CPU-GPU nature, which is adequately exploited in this work in order to achieve real-time unmixing results. The proposed real-time unmixing chain is validated using a variety of hyperspectral scenes. The remainder of the paper is organized as follows. Section II describes the different algorithms used to implement the unmixing chain. Section III describes a hybrid CPU-GPU parallel implementation of the unmixing chain and a multi-core CPU implementation. Section IV presents experimental results using a variety of hyperspectral scenes. Section V concludes the paper with some remarks and hints at plausible future research lines.

[1] https://developer.nvidia.com/cuBLAS (accessed April 2015)
[2] http://openmp.org/wp/ (accessed April 2015)
[3] https://www.gnu.org/software/gsl/ (accessed May 2015)

II. HYPERSPECTRAL UNMIXING CHAIN

Our unmixing chain is made up of three main algorithms: HySime for estimating the number of endmembers, VCA for estimating the endmember signatures, and unconstrained abundance estimation. In the following we describe these algorithms.

A. HySime Algorithm

The noise estimation required by this phase has been carried out assuming additive noise, as shown by (1). A hyperspectral image S can be represented as a LxN matrix, containing N spectral vectors of L dimensions. Each column of this matrix, denoted hereinafter as z_i, is the data associated to the i-th band. We can define the matrix $Z_{\partial_i} = [z_1,...,z_{i-1},z_{i+1},...,z_L]$, and assume that each vector z_i is a linear combination of the remaining L-1 bands. Those vectors can be written as:

$$z_i = Z_{\partial_i}\beta_i + \xi_i \quad (2)$$

where β_i is an L-1 dimensional regression vector and ξ_i is the N-1 dimensional modeling error vector. The least squares estimator of β_i, with $i \in \{1,...,L\}$, is given by:

$$\hat{\beta}_i = (Z_{\partial_i}^T Z_{\partial_i})^{-1} Z_{\partial_i}^T z_i \,, \quad (3)$$

and the noise estimation is given by:

$$\hat{\xi}_i = z_i - Z_{\partial_i}\hat{\beta}_i \,. \quad (4)$$

Now, the expression of the noise correlation matrix $\hat{S}_n = [\hat{\xi}_1,...,\hat{\xi}_N]^T[\hat{\xi}_1,...,\hat{\xi}_N]/N$ can be obtained. The pseudocode of the algorithm used for noise estimation is given below.

ALGORITHM 1 - NOISE ESTIMATION FOR HYSIME

INPUT $S \equiv [s_1, s_2, ..., s_N]$
1: $Z := S^T$, $\hat{S} = (Z^T Z)$;
2: $S' := \hat{S}^{-1}$;
3: **for** i := *1* **to** L **do**
4: $\hat{\beta}_i := \left([S']_{\partial_i,\partial_i} - \frac{[S']_{\partial_i,i}[S']_{i,\partial_i}}{[S']_{i,i}} \right) [\hat{S}]_{\partial_i,i}$;
5: $\hat{\xi}_i := z_i - Z_{\partial_i}\hat{\beta}_i$;
6: **end for**
OUTPUT $\hat{\xi}$

The notation $[S']_{\partial_i,\partial_i}$ represents the matrix S' after removing the i-th row and the i-th column, while $[S']_{i,\partial_i}$ and $[S']_{\partial_i,i}$ represent, respectively, the i-th row of the matrix $[S']_{:,\partial_i}$ and the matrix $[S']_{i,\partial_i}^T$. After this preliminary step, the dimensionality of the hyperspectral subspace can be estimated. This estimation is based on the identification of a set of orthogonal directions. The subspace spanned by the signal is determined by seeking the minimum square error between the

original signal x and a noisy projection obtained by the vector $y = x + n$, where n is a zero-mean Gaussian distributed noise. The subspace is determined by minimizing a two-termed objective function, made up by the power of the signal projection error and the power of noise projection. The first term is a decreasing function of the subspace dimension, while the latter is an increasing one. This leads to this expression:

$$(\hat{p},\hat{\pi}) = \arg\min_{p,\pi}\left\{ c + \sum\nolimits_{j=1}^{k} (\underbrace{-b_{i_j} + 2\sigma^2_{i_j}}_{\delta_{i_j}}) \right\}, \qquad (5)$$

where π is the permutation of indices $i = 1,\dots,L$ and k is the subspace dimension. b_{i_j} and $\sigma^2_{i_j}$ are quadratic forms given by:

$$b_{i_j} = e^T_{i_j}\hat{S}_y e_{i_j} \;,\; \sigma^2_{i_j} = e^T_{i_j}\hat{S}_n e_{i_j} \qquad (6\text{-}7)$$

where e_i are the eigenvalues of the estimated signal correlation matrix $\hat{S}_x$ [2]. The pseudocode of the algorithm used for subspace estimation is given in Algorithm 2.

ALGORITHM 2 - SIGNAL SUBSPACE ESTIMATION FOR HYSIME

INPUT $S \equiv [s_1, s_2, \dots, s_N]$, $\hat{S}_y = SS^T / N$, $\hat{\xi}$
1: $\hat{S}_n := \frac{1}{N}\sum_i \left(\hat{\xi}_i \hat{\xi}_i^T\right)$;
2: $\hat{S}_x := \frac{1}{N}\sum_i \left[\left(s_i - \hat{\xi}_i\right)\left(s_i - \hat{\xi}_i^T\right)\right]$;
3: $E := [e_1, \dots, e_L]$; { e_i are the eigenvalues of $\hat{S}_x$ }
4: $\delta := [\delta_1, \dots, \delta_L]$;
5: $(\hat{\delta}, \hat{\pi}) := sort(\delta)$; { δ_i sorted in ascending order}
6: $\hat{p} :=$ number of terms $\delta_i < 0$;
OUTPUT $\hat{p}$, $\hat{X} = \left\langle \left[e_{\hat{i}_1}, \dots, e_{\hat{i}_{\hat{p}}} \right] \right\rangle$; {signal subspace}

B. VCA algorithm

The VCA is based on two geometrical assumptions: i) the endmembers are the vertices of a simplex, and ii) the affine transformation of a simplex is also a simplex. The algorithm assumes that the endmembers appear in pure form in the scene. The set $\{a \in R^p : 1^T a = 1, a \ge 0\}$ is a simplex, so by using ii) we deduce that the set $\{s \in R^N : x = Ma, 1^T a = 1, a \ge 0\}$ is also a simplex. Assuming that $n = 0$ in (1), all the spectral pixels belong to the convex cone $C = \{s \in R^N : s = M\gamma a, 1^T a = 1, a \ge 0, \gamma \ge 0\}$, where γ is a scale factor given by, for example, illumination variability due to surface topography. Projecting the simplex onto a hyperplane $s^T u = 1$ allows us to obtain the new simplex $\{y \in R^N : y = s / (s^T u), s \in C\}$. It is important to choose a suitable u that ensures that no observed vectors are orthogonal to it. The algorithm iteratively performs this projection onto the direction orthogonal to the subspace spanned by the endmembers already determined, and the extreme of the projection in each iteration is the new endmember. In other words, VCA projects data onto the subspace signal with dimensionality $\hat{p} \ll N$, in order to decrease computational complexity. In our case, the dimensionality is estimated in the first step of the unmixing chain (using the HySime algorithm).

ALGORITHM 3 - VCA

INPUT $S \equiv [s_1, s_2, \dots, s_N]$, $\hat{p}$
1: $SNR_{th} = 15 + \log_{10}(\hat{p})$;
2:**if** $SNR > SNR_{th}$ **then**
3: $d := \hat{p}$;
4: $X := U_d^T S$; { U_d obtained by SVD of SS^T / N }
5: $u := mean(X)$; {mean of each row}
6: $[Y]_{:,j} := [X]_{:,j} / ([X]_{:,j} u)$;
7: **else**
8: $d := \hat{p} - 1$;
9: $[X]_{:,j} := U_d^T([S]_{:,j} - \bar{s})$; { U_d obtained by PCA}
10: $h := \arg\max_{j=1,\dots,N} \left\|[X]_{:,j}\right\|$;
11: $H := [h \,|\, h \,|\, \dots \,|\, h]$; { H is a 1xN vector}
12: $Y := \begin{bmatrix} X \\ H \end{bmatrix}$;
13:**end if**
14: $e_u = [0, \dots, 0, 1]^T$;
15: $A := [e_u \,|\, 0 \,|\, \dots \,|\, 0]$; { A is a $\hat{p}$ x $\hat{p}$ matrix}
16:**for** $i := 1$ **to** $\hat{p}$ **do**
17: $w := rand(0, I_{\hat{p}})$; { w is a zero-mean Gaussian vector of covariance $I_{\hat{p}}$ }
18: $f := ((I - (AA^{\bullet})w) / \left(\left\|(I - AA^{\bullet})w\right\|\right)$;
19: $v = f^T Y$;
20: $g := \arg\max_{j=1,\dots,N} \left\|[v]_{:,j}\right\|$
21: $[A]_{:,i} := [Y]_{:,g}$;
22: $[index]_i := g$;
23:**end for**
24: **if** $SNR > SNR_{th}$ **then**
25: $\hat{M} := U_{\hat{p}}[X]_{index}$;
26:**else**
27: $\hat{M} := U_{\hat{p}}[X]_{index} + \bar{s}$;
28:**end if**
OUTPUT $\hat{M}$; {estimated mixing matrix}

The projection can be performed using *singular value decomposition* (SVD) or *principal component analysis* (PCA). According to [3], we made this choice using a threshold based on signal-to-noise ratio (SNR). In [3] it has been empirically shown that, when the data are noiseless, an SVD followed by a projection gives the best results, while in the other case a PCA generally performs better. It must be noted that in [3] the threshold value (SNR_{th}) for determining the dimension of subspace for data projection is also defined, that is $\hat{p}$ if the SNR is higher than SNR_{th} or $\hat{p} - 1$ otherwise. The pseudocode of the VCA is shown in Algorithm 3.

At this point, it is important to note that the noise estimation performed by VCA is the same one described for HySime. Of course, by connecting the two algorithms it only needs to be performed once. Lines 1-13 in Algorithm 3 are related to dimensionality reduction, while lines 14-23 are core of VCA algorithm. The remaining lines are related to the estimation of the mixing matrix.

C. Unconstrained Abundance Estimation

For the abundance estimation part of our unmixing chain, we use unconstrained abundance estimation. This part is based on removing the noise term from equation (1), so the hyperspectral image is expressed as:

$$S = Ma, \tag{8}$$

where M is the mixing matrix estimated by VCA and a is the abundance matrix. The pixel reconstruction error is:

$$e = \left\| S - \hat{M}a \right\|^2 . \tag{9}$$

The error is minimized by finding a suitable a, and the least square solution is given by:

$$a = (M^T M)^{-1} M^T S, \tag{10}$$

where $(M^T M)^{-1} M^T$ is the pseudo-inverse of matrix M.

III. Unmixing Chain Parallel Implementation

For developing the parallel implementation of the unmixing chain described in Section II, we started from the Matlab implementations of HySime[4] and VCA[5] which are available online. We verified the results of our parallel implementation using the outputs provided by the two original implementations. Our implementation does not change the outputs of each single algorithm. The Matlab version has been used in order to identify intensive code parts. We profiled the code using different synthetic hyperspectral images generated using a suitable Matlab script [2] that chooses endmember signatures from USGS Library (1995 version)[6]. The synthetic images are different in both size and number of endmembers; the size ranges from 2000 to 500000 pixels, while the number of endmembers ranges from 15 to 25. Matrix-matrix operations and matrix-vector operations were identified as the most intensive code parts. Those calculations are required in the noise estimation stage and in the computation of $\hat{S}_n$ and $\hat{S}_x$ for signal subspace estimation, and also in the dimensionality reduction and the computation of vector v for the identification of endmembers. In order to obtain the best possible execution times, we parallelized all those calculations using GPUs. The main limitation of this kind of platform is the need for intensive memory transfers, which are used for copying data from the CPU RAM memory to GPU global (video) memory. The communication is performed by a PCI Express bus, so it is important to minimize the amount of transferred data. Moreover, only big amounts of data should be moved to the GPU in order to quickly perform different calculations and finally transfer the results back to the CPU.

For the signal subspace estimation, we transferred the image to the GPU memory at the beginning, using the *cudaMemcpy* function specifying the HostToDevice parameter. After that we performed the matrix multiplication using the function *cublasDgemm*, which executes the calculation:

$$C = \alpha op(A) op(B) + \beta C, \tag{11}$$

where A, B, C are matrices and α, β are scalars. The notation $op(\bullet)$ indicates the transpose operation. In our case, we set the scalar β to zero and the scalar α to 1 or to $1/N$, as appropriate.

At this point, it is important to note that we chose to manually manage the memory transfers in order to obtain better performances. The CuBLAS library functions used in this work are only highly optimized GPU kernels. CuBLAS stores matrices in column-major order, while standard C matrices are stored in row-major order, so it is important to convert data from the first schema to the latter. Obviously, for converting a matrix from row-major to column-major and vice-versa, the data must be transposed. Since transposition of a big matrix requires time, we choose to implement this operation using the GPU too. For that purpose, we used the function *cublasDgeam*, which performs the calculation:

$$C = \alpha op(A) + \beta op(B). \tag{12}$$

For transposing matrix A, we set the parameters $\alpha = 1$ and $\beta = 0$ and chose a suitable value to $op(A)$. It is important to note that the kernel ignores the value of matrix B when $\beta = 0$. Concerning other operations such as eigenvalues (δ) sorting, or negative values counting, we chose to use the CPU instead of the GPU, since the amount of data is lower. For the eigenvalues calculation, we implemented an optimized algorithm based on Householder reductions and QR transformations, while for the sorting step we implemented a *quicksort* algorithm. The other operations have been implemented using OpenMP, specifically using the statement *#pragma omp parallel for*, which allocates a loop iteration to each thread. It is possible to use this strategy since the algorithm iterations are independent in our case. Moreover, we specify which data are shared among threads and which data are private using the suitable options. We chose to share input and output values among threads, while loop management variables are private to each thread. Finally we decided to use a static scheduling rather than a dynamic one, because we empirically observed that in our experiments this option performs better. In some cases we also used the *reduction* clause to accumulate results of various threads in a single variable. For example, the final count of negative values is performed in parallel using a reduction as follows: each thread works on a chunk of data, and finally the master thread accumulates the final results of each thread in a single variable. All the calculations needed by the signal subspace estimation step have been performed using double precision floating point, to guarantee the same precision as Matlab. However, the VCA step requires single precision floating

[4] http://www.lx.it/pt/~bioucas/code/demo_Hysime.zip (accessed May 2015)
[5] http:// www.lx.it/pt/~bioucas/code/demo_vca.zip (accessed May 2015)
[6] http://speclab.cr.usgs.gov (accessed May 2015)

point accuracy, so we wrote an OpenMP routine for converting the image and the noise correlation matrix into that format.

All the dimensional reduction operations are highly parallelizable, so the matrix multiplications have been performed using CuBLAS, while the other operations have been performed using OpenMP. The used kernels are the counterpart of the ones described before, with the only difference that the data are represented in single precision floating point, using *cublaSgemm* (for matrix multiplications) and *cublasSgeam* (for matrix transpositions). This transposition cannot be avoided, since we need the results also for the host computation. Concerning the VCA main *for* loop, we transfer the Y matrix only before the loop and store it in the GPU memory, since it is not modified during the various iterations. Each loop iteration then computes the w and f vectors using OpenMP. The f vector is moved to the GPU memory and the product $f^T Y$ is computed using the CuBLAS kernel *cublasSgemv,* which performs the operation:

$$c = \alpha op(A)x + \beta c, \tag{13}$$

where c and x are vectors, α and β are scalars and A is a matrix that can be transposed. We set the parameters $\alpha = 1$ and $\beta = 0$, and finally we transpose the matrix. After this kernel is completed, we need to search for the position of the maximum; this operation is performed by using another CuBLAS kernel called *cublasIsamax*. Since in CuBLAS vectors are stored using *1*-indexing while in C are stored using *0*-based indexing, it is necessary to subtract 1 from the index returned by the kernel for results consistency. Also, it is important to perform the maximum search on the GPU because this avoids to transfer vector v to the CPU, allowing us to improve the execution times even further. All the remaining operations are performed using OpenMP. This computation is schematized in figure 1. Regarding the abundance estimation step, the pseudo-inverse calculation has been performed using OpenMP, since the dimensions of the data involved are not extremely big. The final multiplication in (10) has been performed on the GPU using a suitable kernel provided by the *cublasDgemm* function. SVD has been solved using QR factorization implemented through Householder reflections, while matrix inversion has been solved using LU factorization. Both algorithms have been parallelized with OpenMP. Concerning the BLAS implementation, we exchanged the CuBLAS functions calls and the data transfers with suitable functions, such as *gsl_blas_dgemm*, *gsl_blas_dgeam* and *gsl_blas_sgemm*.

IV. Experimental Results

A. GPU architecture

The NVidia GPU architecture used in this work is codenamed *Kepler*, and includes the new SMX processors architecture and an enhanced memory subsystem, offering better performance in terms of bandwidth [20]. Each SMX unit features 192 fully pipelined floating-point and integer arithmetic CUDA cores. SMX also has special function units (SFUs), for approximated transcendental functions computation. Parallel threads are scheduled in groups of 32 (*warps*) and each SMX has four warp schedulers and eight dispatch units. The most important aspect of this architecture is the possibility to pair double precision instructions with other instructions. Each SMX has 64 KB of on-chip memory that can be configured as 16 KB of shared memory and 48 KB of level 1 (L1) cache or 48 KB of shared memory and 16 KB of L1 cache or 32 KB of shared memory and 32 KB of L1 cache. The shared memory bandwidth for 64 bit and larger load operations is 256 Bytes per core clock. Moreover, there is also a 48 KB read-only L1 cache, directly managed by the compiler. The L2 cache is the primary point of data unification between the SMX for high speed data sharing. Kepler features 1536 KB of L2 cache. Notice that L2 cache allows communication between SMX and GPU global memory (DRAM), which is the main communication channel with the CPU.

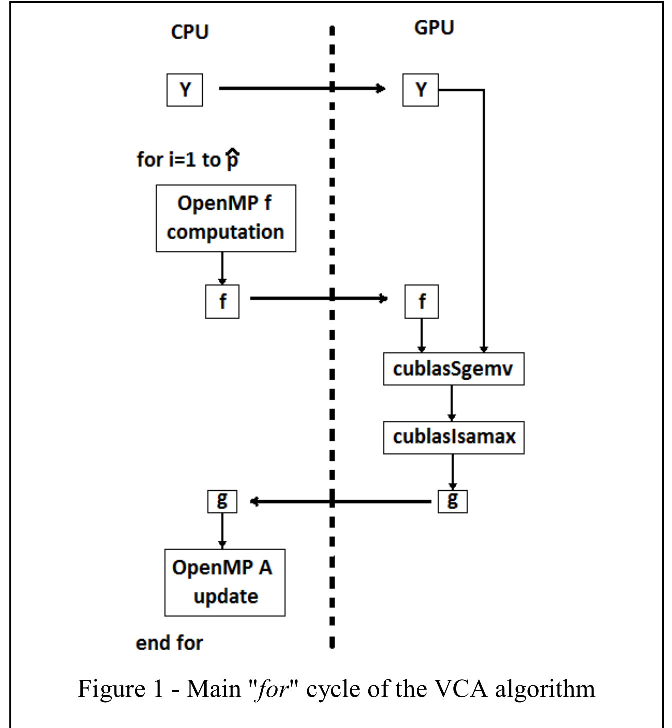

Figure 1 - Main "*for*" cycle of the VCA algorithm

TABLE I
HARDWARE SPECIFICATIONS

	Number of cores	Clock frequency [GHz]	RAM [GB]
CPU Intel 3770 i7	4	3.40	8
Nvidia Tesla K40 active	2880	0.875	12

TABLE II
EXECUTION TIMES FOR SYNTHETIC HYPERSPECTRAL IMAGES. IN BRACKETS THERE IS THE SPEED UP WITH RESPECT TO MATLAB VERSION

Pixel number	Bands	$\hat{p}$	MATLAB [s]	BLAS [s]	CUDA and OpenMP [s]
2000	224	15	0.430	0.390 (1.10x)	0.412 (1.04x)
10000	224	15	0.894	0.785 (1.13x)	0.515 (1.73x)
20000	224	15	1.735	1.320 (1.31x)	0.615 (2.82x)
50000	224	15	3.996	2.125 (1.88x)	0.920 (4.34x)
75000	224	20	4.528	3.246 (1.39x)	1.173 (3.86x)
100000	224	20	6.327	5.426 (1.16x)	1.499 (4.22x)
200000	224	20	15.421	9.360 (1.64x)	2.563 (6.01x)
500000	224	25	32.963	29.250 (1.12x)	5.856 (5.63x)

The Kepler register files, shared memories, L1 and L2 caches and DRAM memory are protected by Single-Error Correct Double-Error Detect (SECDED) Error-Correction Code (ECC). Moreover, read-only L1 cache performs single-error correction through a parity check.

B. Performance Evaluation

We generated different synthetic hyperspectral images with increasing sizes (from 2000 pixels to 500000 pixels) choosing endmembers signatures from the USGS Library using the MATLAB script provided in [10]. Those images have been used for two main purposes: the first one is to compare the results of the MATLAB implementation and our parallel implementation, while the second reason is to be able to characterize the computational complexity of the whole unmixing chain. We also used real images for evaluation purposes: two well-known AVIRIS images collected over the Cuprite mining district (47750 pixels, 188 bands) and the World Trade Center area (314368 pixels, 224 bands); a Hydice image collected over a forest radiance environment (4096 pixels, 169 bands); and a ROSIS hyperspectral image collected over the city of Pavia, Italy (1024000 pixels, 102 bands). The tests have been performed on a PC equipped with an Intel 3770 i7 CPU, working at 3.40 GHz, with 8 GB of RAM. The used GPU is a Tesla K40 active equipped with the previously described Kepler architecture (Table I). The communication between the CPU and the GPU is performed by a PCI-e 2.0 16x bus. The operating system is a 64 bit Ubuntu 12.10, which is fully supported by the CUDA 6.0 environment. The code compilation is performed using *nvcc* compiler for GPU code and gcc for CPU code. We optimized the code using suitable compilation flags such as *-O3* and *-fopenmp* for the CPU code. Table II shows the execution times of the MATLAB implementation, our BLAS implementation and our hybrid CUDA and OpenMP parallel implementation using synthetic images. We measured execution times using MATLAB profiler for the MATLAB implementation and suitable functions such as *omp_get_wtime* for the other implementations. It is important to note that our implementation outperforms MATLAB and BLAS as the image dimension grows up, and since AVIRIS acquires about 300000 pixels in 5 seconds, the real-time constraint is satisfied. Moreover, it is possible to identify the image sizes that make the hybrid parallel approach convenient with respect to the pure CPU multi-core one. It must be noticed that, in our case, the hybrid solution performs better since we have images with 10000 pixels, which is a small size when compared with standard hyperspectral image sizes. The real-time compliance assertion is also supported by experiments conducted using real images, which are reported in Table III. Specifically, it must be noted that the Pavia image is spatially four times bigger than a regular AVIRIS scene (unless it has less bands, but this is due to the different instrument used for acquisition), so the execution time is about 20 seconds. The obtained execution times fully meet real time constraints also in this particular case. All the reported times (except for the MATLAB ones) are obtained as the mean of different executions with standard deviation always less than 2%. We also evaluated accuracy of the obtained unmixing results with respect to the serial implementation. In all cases the results have a mean square error difference that is less than 1%.

TABLE III
EXECUTION TIMES FOR REAL HYPERSPECTRAL IMAGES

Image	Pixel number	Bands	$\hat{p}$	CUDA and OpenMP [s]
Hydice	4096	169	18	0.188
Cuprite	47750	188	18	0.631
WTC	314368	224	23	3.625
Pavia	1024000	102	58	5.312

If we compare the obtained results with other developments in the literature, such as the GPU implementation of the pixel purity index (PPI) algorithm [12] and other geometrical techniques [13], we can outline the following aspects. In [12] tests are conducted on the AVIRIS World Trade Center image, implementing a parallel unmixing chain on different architectures such as a Beowulf cluster at NASA's Goddard Space Flight Center, an heterogeneous network made up 16 workstations mounting Intel Xeon, AMD Athlons and SUNW UltraSparc processors, a Xilinx Virtex-II FPGA, and a NVIDIA Tesla C1060 GPU. The best result is obtained by the Beowulf cluster, which took 4.98 s to perform the full unmixing process, using 256 Intel Xeon nodes working at 2.4 GHz. Our execution time is lower than this one. Moreover, it must be highlighted that we are using a desktop system, which is much less expensive in terms of cost and maintenance. In

[13] tests have been performed using the AVIRIS Cuprite image, using a system made up of two Quad Core Intel Xeon working at 2.53 GHz with 12 physical cores, a NVIDIA GeForce GTX 580 GPU and a NVIDIA Tesla C1060 GPU. The best results were obtained by the first GPU, which took about 0.6 seconds to perform calculations. This time is very close to ours, but we implemented more recent unmixing techniques, thus achieving a better precision. In other works, parallel implementations of HySime and VCA have been presented [15, 16]. The main difference with our work is that we developed an optimized version for a more recent GPU architecture. Moreover, we developed a hybrid parallel application using both CUDA (GPU) and OpenMP (multi-core), while in [15] only GPU computing was used. The memory transfers have also been optimized, exploiting level 1, 2 and 3 CuBLAS functions and trying to perform most of the calculations in the GPU. For example, the core of the VCA algorithm has been optimized in terms of finding the minimum value of a vector. This value is computed directly on the GPU using a suitable kernel, so we only need to transfer an integer value to the CPU. The memory transfers took from 20% of the execution time to 60% (from 0.1 seconds to 3.2 seconds), according to image dimension. Finally, we have experimental data that enable us to characterize the computational complexity of the whole algorithm. In particular, using Nvidia Visual Profiler on our hybrid implementation with the WTC image, we found that 35% of the execution time is used for memory transfers. The remaining GPU time is used mainly for the *cublasDgemm* and the *cublasSgemm* that occupy, respectively, 25% and 15% of the total execution time. To conclude this section, we emphasize that the main limitation of employing GPU technology in spaceborne missions is related to power consumption, since a GPU like the one used in this study has a power consumption of up to 200 W. This issue will be probably solved in future GPU generations, since limiting power consumption is an important objective of GPU vendors such as NVidia for allowing the scientific community to use their devices in embedded applications. Moreover, memory communication is a well-known performance bottleneck, but this problem could be solved using a system where the memories of the CPU and GPU are unified.

V. Conclusions and Future Lines

In this paper we presented a hybrid CPU-GPU parallel implementation of a hyperspectral unmixing chain made up of the HySime algorithm for estimating the number of endmembers, the VCA for identifying the endmember signatures, and unconstrained abundance estimation. The parallel implementation exploits hybrid parallelization, using both CUDA and OpenMP. The parallel implementation has been tested on different synthetic image sizes, on an Intel i7 CPU connected with a NVIDIA Kepler GPU. The parallel unmixing chain fully meets real-time constraints, since a regular AVIRIS scene acquired in 5 seconds is processed in about 3.6 s. As future research we are planning on performing optimizations of other hyperspectral analysis algorithms for classification and compression using a hybrid CPU-GPU framework such as the one presented in this contribution.

Acknowledgment

The authors gratefully acknowledge NVidia Corporation for the donation of the NVidia Kepler GPU used for this research.

References

[1] R. O. Green, M. L. Eastwood, C. M. Sarture, T. G. Chrien, M. Aronsson, B. J. Chippendale, J. A. Faust, B. E. Pavri, C. J. Chovit, M. Solis, M. R. Olah and O. Williams, "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer," *Remote Sens. Environ.*, 65(3), pp. 227-248, Sep. 1998.

[2] T. Stuffler, K. Förster, S. Hofer, M. Leipold, B. Sang, H. Kaufmann, B. Penné, A. Mueller and C. Chlebek, "Hyperspectral imaging - An advanced instrument concept for the EnMAP mission," *Acta Astronaut.*, 65(7/8), pp. 1107-1112, Oct./Nov. 2009.

[3] J. S. Pearlman, P. S. Barry, C. C. Segal, J. Shepanski, D. Beiso and S. L. Carman, "Hyperion, a space-based imaging spectrometer," *IEEE Trans. Geosci. Remote Sens.*, 41(6), pp. 1160-1173, Jun. 2003.

[4] N. Keshava and J. F. Mustard, "Spectral unmixing," *IEEE Signal Process. Mag.*, 19(1), pp. 44-57, Jan. 2002.

[5] J. M. Bioucas-Dias, A. Plaza, N. Dobigeon, M. Parente, Q. Du, P. Gader, J. Chanussot, "An overview on hyperspectral unmixing: Geometrical, statistical, and sparse regression based approaches," *IEEE J. Sel. Topics Appl. Earth Observ. Rem. Sens.*, 5(2), pp. 354-379, 2012.

[6] R. Heylen, M. Parente and P. Gader, "A review of nonlinear hyperspectral unmixing methods," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sensing*, vol. 7, no. 6, pp.1844-1868, 2014.

[7] L. L. Scharf, "Statistical Signal Processing, Detection Estimation and Time Series Analysis," Reading, MA, Addison-Wesley, 1991.

[8] J. Bruske and G. Sommer, "Intrinsic dimensionality estimation with optimally topology preserving maps," *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(5), pp. 572-575, May 1998.

[9] C. Chang and S. Wang, "Constrained band selection for hyperspectral imagery," *IEEE Trans. Geosci. Rem. Sens.*, 44(6), pp. 1575-1585, June 2006.

[10] J. M. P. Nascimento and J. M. Bioucas-Dias, "Hyperspectral subspace identification," *IEEE Trans. Geosci. Remote Sens.*, 46(8), pp. 2435-2445, Aug. 2008.

[11] A. Plaza, J. Plaza, A. Paz and S. Sanchez, "Parallel hyperspectral image and signal processing," *IEEE Signal Process. Mag.*, vol 28, no. 3, pp. 119-126, 2011.

[12] A. Plaza, Q. Du, Y. L. Chang, R. L. King, “High Performance Computing for Hyperspectral Remote Sensing,” *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, 4(3), pp. 528-544, Sept. 2011.

[13] S. Bernabé, S. Sanchez, A. Plaza, S. López, 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.*, 6(3), pp. 1386-1398, 2013.

[14] C. I. Chang and Q. Du, "Estimation of number of spectrally distinct signal sources in hyperspectral imagery," *IEEE Trans. Geosci. Remote Sens.*, 42(3), pp. 608-619, 2004.

[15] S. Bernabé, S. López, A. Plaza and R. Sarmiento, "GPU implementation of an automatic target detection and classification algorithm for hyperspectral image analysis," *IEEE Geosci. And Remote Sens. Letters*, 10(2), pp. 221-225, 2013.

[16] C. I. Chang, "Hyperspectral Data Processing: Algorithm Design and Analysis", *John Wiley & Sons Inc*, 2013

[17] J. M. P. Nascimento and J. M. Bioucas-Dias, "Vertex component analysis: A fast algorithm to unmix hyperspectral data," *IEEE Trans. Geosci. Remote Sens.*, 43(4), pp. 898-910, Apr. 2005.

[18] E. Torti, M. Acquistapace, G. Danese, F. Leporati and A. Plaza, "Real-Time Identification of Hyperspectral Subspaces," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, 7(6), pp. 2680-2687, 2014.

[19] A. Barberis, G. Danese, F. Leporati, A. Plaza and E. Torti, "Real-Time Implementation of the Vertex Component Analysis Algorithm on GPUs," *IEEE Geosci. and Rem. Sens. Letters*, 10(2), pp. 251-255, 2013.

[20] NVIDIA Corporation, "NVIDIA's Next Generation CUDA™ Compute Architecture: Kepler GK110", *Whitepaper available online*.