Parallel Hyperspectral Unmixing on GPUs

José M. P. Nascimento, José M. Bioucas-Dias, *Member, IEEE*, José M. Rodríguez Alves, Vítor Silva, and Antonio Plaza, *Senior Member, IEEE*

***Abstract*—This letter presents a new parallel method for hyperspectral unmixing composed by the efficient combination of two popular methods: vertex component analysis (VCA) and sparse unmixing by variable splitting and augmented Lagrangian (SUNSAL). First, VCA extracts the endmember signatures, and then, SUNSAL is used to estimate the abundance fractions. Both techniques are highly parallelizable, which significantly reduces the computing time. A design for the commodity graphics processing units of the two methods is presented and evaluated. Experimental results obtained for simulated and real hyperspectral data sets reveal speedups up to 100 times, which grants real-time response required by many remotely sensed hyperspectral applications.**

***Index Terms*—Graphics processing unit (GPU), parallel methods, sparse unmixing by variable splitting and augmented Lagrangian (SUNSAL), unsupervised hyperspectral unmixing, vertex component analysis (VCA).**

I. Introduction

REMOTE hyperspectral sensors collect hundreds of images within their ground instantaneous field of view corresponding to nearly contiguous spectral channels of high spectral resolution [1]. This technology provides enough spectral resolution for material identification, facilitating an enormous number of applications in the fields of military surveillance, target detection, environmental monitoring, detection of oil spills and other types of chemical contamination, biological hazards prevention, and food safety [1], many of which require real-time or near-real-time response [2].

Due to the generally low spatial resolution provided by these devices and the natural composition of the terrestrial surface, each pixel is generally a mixture of several spectrally distinct materials (also called *endmembers*) [1], [3]. Hyperspectral unmixing is a source separation problem which amounts to estimating the number of endmembers, their spectral signatures, and their abundance fractions (i.e., the percentage of each endmember) [1].

Considering the linear mixture model, each pixel denoted by $\mathbf{y} \in \mathbb{R}^L$ (L is the number of bands) is given by $\mathbf{y} = \mathbf{M}\mathbf{s} + \mathbf{w}$, where $\mathbf{M} \equiv [\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_p]$ is a full-rank $L \times p$ mixing matrix ($\mathbf{m}_j$ denotes the jth endmember signature), p is the number of endmembers present in the covered area (with $p < L$), $\mathbf{s} = [s_1, s_2, \ldots, s_p]^{\mathrm{T}}$ is the abundance vector containing the fractions of each endmember, and $\mathbf{w}$ is the additive noise vector (notation $(\cdot)^{\mathrm{T}}$ stands for vector transposed).

To fix the notation, let $\mathbf{Y} \equiv [\mathbf{y}_1, \ldots, \mathbf{y}_N] \in \mathbb{R}^{L \times N}$ denote a matrix holding the observed spectral vectors, $\mathbf{S} \equiv [\mathbf{s}_1, \ldots, \mathbf{s}_N] \in \mathbb{R}^{p \times N}$ a matrix holding the respective abundance fractions, and $\mathbf{W} \equiv [\mathbf{w}_1, \ldots, \mathbf{w}_N] \in \mathbb{R}^{L \times N}$ a matrix accounting for additive noise. Therefore, the observed data set is given by

$$\mathbf{Y} = \mathbf{M}\mathbf{S} + \mathbf{W}. \tag{1}$$

Due to the nature of the acquisition process, at a given pixel, abundance fractions sum to 1 and are nonnegative (the so-called *abundance sum constraint* and *abundance nonnegativity constraint*). This abundance fraction dependency prohibits the use of canonical source separation methods for hyperspectral unmixing [3]. However, considering these constraints, abundance fractions are in the following $p-1$ probability simplex:

$$\left\{\mathbf{S} \in \mathbb{R}^{p \times N} : \mathbf{S} \succeq 0, \mathbf{1}_p^{\mathrm{T}}\mathbf{S} = \mathbf{1}_n^{\mathrm{T}}\right\} \tag{2}$$

where $\mathbf{S} \succeq 0$ means $s_{ij} \geq 0$ for $i = 1, \ldots, p$ and $j = 1, \ldots, N$, and $\mathbf{1}_p$ and $\mathbf{1}_N$ respectively denote a $1 \times p$ and a $1 \times N$ column vector filled with ones.

Considering that the columns of $\mathbf{M}$ are affinely independent, the observed spectral vectors for a given scene belong to a $p-1$ simplex in $\mathbb{R}^L$ whose vertices correspond to the endmembers. This property has been used in recent years by several approaches to perform hyperspectral unmixing [1]. Among these methods, also called geometrical-based approaches, there are some efficient algorithms, from the computational point of view, that assume the presence of pure pixels in the data set, meaning there is at least one spectral vector on each vertex of the data simplex [1], [4]. Some popular algorithms taking this assumption are the following: the *vertex component analysis* (VCA), [5], the *automated morphological endmember extraction* [6], the *pixel purity index*, [7], the N-FINDR [8], the *alternating volume maximization*, the *successive volume maximization* (SVMAX) [9], and the *robust and recursive nonnegative matrix factorization* (RRNMF) [10]. Usually, these methods are followed by a fully constrained least square (FCLS) estimation or by a maximum likelihood estimation of the abundance fractions to complete the unmixing procedure [11]. Recently, a method called *spectral unmixing by splitting and augmented Lagrangian* (SUNSAL) [12] has been proposed to cope with the abundance fraction estimation. SUNSAL is

Manuscript received February 25, 2013; revised June 24, 2013; accepted July 14, 2013. Date of publication September 20, 2013; date of current version November 25, 2013. This work was supported in part by the Instituto de Telecomunicações and in part by the Fundação para a Ciência e Tecnologia under Project PEst-OE/EEI/LA0008/2013.

J. M. P. Nascimento is with the Instituto de Telecomunicações, 1049-001 Lisbon, Portugal and also with the Instituto Superior de Engenharia de Lisboa, 1959-007 Lisbon, Portugal (e-mail: zen@isel.pt).

J. M. Bioucas-Dias is with the Instituto de Telecomunicações, 1049-001 Lisbon, Portugal and also with the Instituto Superior Técnico, Technical University of Lisbon, 2744-016 Lisbon, Portugal (e-mail: bioucas@lx.it.pt).

J. M. Rodríguez Alves is with the Instituto de Telecomunicações, 1049-001 Lisbon, Portugal (e-mail: josdumper@gmail.com).

V. Silva is with the Institute of Telecommunications, Departamento de Engenharia Electrotécnica e de Computadores, Universidade de Coimbra 3030-290 Coimbra, Portugal (e-mail: vitor@co.it.pt).

A. Plaza is with the Hyperspectral Computing Laboratory, University of Extremadura, 06006 Cáceres, Spain (e-mail: aplaza@unex.es).

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

Digital Object Identifier 10.1109/LGRS.2013.2274328

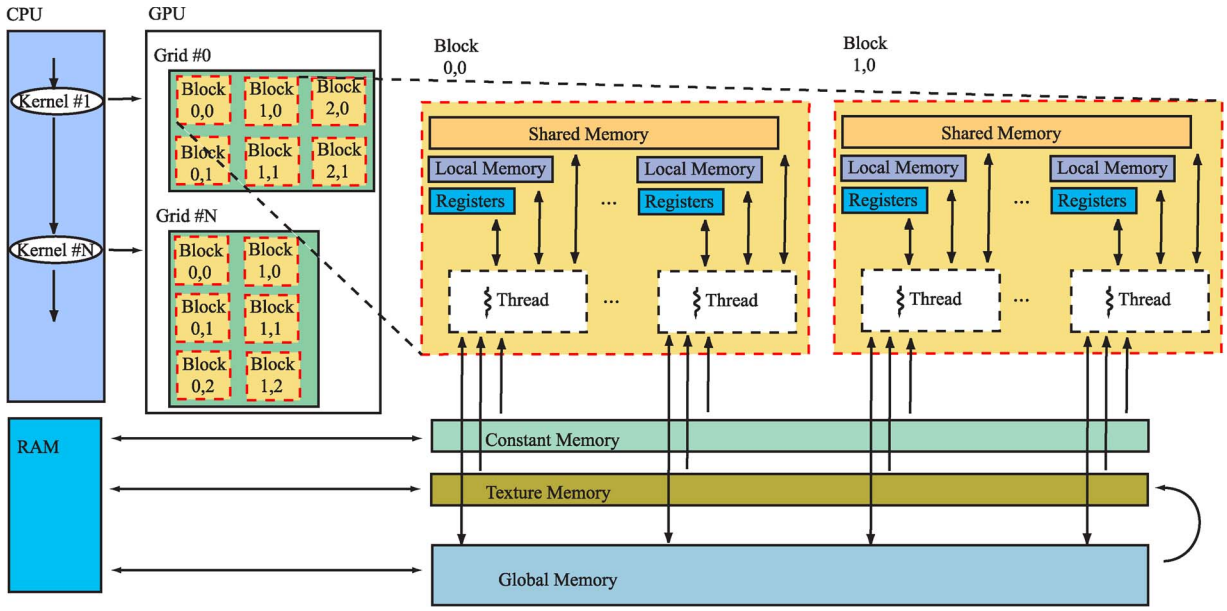

Fig. 1. Typical NVidia GPU architecture, computation, and data transfer flow from/to CPU.

an instance on the alternating direction method of multipliers (ADMM) [13], which decomposes a difficult problem into a sequence of simpler ones, resulting in a very fast method when compared with algorithm FCLS [11].

However, due to the amount of hyperspectral data together with the real-time requirements of several applications, high-performance computing is needed to accelerate hyperspectral imaging algorithms [2]. In recent years, parallel computing techniques have been widely used to accelerate hyperspectral unmixing methods, namely, on graphics processing units (GPUs), due to their extremely high floating-point processing performance, huge memory bandwidth, compact size, and comparatively low cost [14], [15].

This letter proposes a combination of VCA and SUNSAL algorithms to build a very efficient parallel hyperspectral unmixing method. VCA extracts the endmember signatures, and SUNSAL estimates the abundance fractions. Both, VCA and SUNSAL are highly parallelizable. Herein, a parallel solution, using the multicore and multiplatform portable OpenCL programming language, for GPUs is presented. The remainder of this letter is organized as follows. Section II briefly describes the VCA and SUNSAL methods. Section III describes their GPU implementation. Section IV evaluates the proposed parallel implementations in terms of computational complexity and speedup. Section V outlines the main conclusions of the letter with some remarks and future research lines.

II. Unmixing Method

A. VCA Method

VCA is an unsupervised method for unmixing hyperspectral linear mixtures and is based on the geometry of convex sets. It exploits the fact that the endmembers are the vertices of the simplex. The VCA algorithm iteratively projects data onto a direction orthogonal to the subspace spanned by the endmembers already determined. The new endmember signature corresponds to the extreme of the projection. The algorithm iterates until all endmembers are exhausted. A similar structure to VCA can be found in the SVMAX and RRNMF methods. Algorithm 1 shows the main steps of the VCA, where symbol $\widehat{\mathbf{M}}$ denotes the estimated mixing matrix and $[\widehat{\mathbf{M}}]_{:,j}$ stands for the jth column of $\widehat{\mathbf{M}}$. The red and yellow lines denote operations to be computed in CPU and GPU, respectively.

B. SUNSAL Method

The endmember's abundance estimation problem can be posed in the framework of convex optimization. Under this context, the ADMM [13] is a powerful algorithm that can be parallelized. The SUNSAL method is an ADMM-based approach to estimating the abundance fractions under *sum one* and *nonnegativity* constraints. Assuming that matrix $\mathbf{M}$ is obtained by the VCA algorithm, the abundance fraction estimation can be defined as

$$\begin{aligned} &\min_{\mathbf{S}} \frac{1}{2}\|\mathbf{Y}-\mathbf{M}\mathbf{S}\|_F^2 \\ &\text{subject to : } \mathbf{S} \succeq 0, \mathbf{1}_p^{\mathrm{T}}\mathbf{S} = \mathbf{1}_n^{\mathrm{T}} \end{aligned} \tag{3}$$

where notation $\|(\cdot)\|_F$ stands for the Frobenius norm. The aforementioned formulation is a particular case of the constrained $\ell_2 - \ell_1$ problems solved by SUNSAL corresponding to the absence of the ℓ_1 term. The pseudocode is presented in Algorithm 2.

III. GPU Implementations

The typical GPU architecture is organized into an array of highly threaded streaming multiprocessors (SMs), where each multiprocessor is characterized by a single instruction multiple data architecture, i.e., in each clock cycle, each processor executes the same instruction while operating on multiple data streams. Each SM has a number of streaming processors that share a control logic and instruction cache and have access to a local shared memory and to local cache memories in the multiprocessor, while the multiprocessors have access to the global GPU (device) memory. GPUs can be abstracted in terms of a stream model, under which all data sets are represented as streams. Fig. 1 presents a typical architecture and the data flow communication between CPU and GPU.

TABLE I
VCA AND SUNSAL PROCESSING TIME (10^{-3} s) FOR AN INTEL CORE i7-2600 CPU AS A FUNCTION OF THE NUMBER OF ENDMEMBERS (p) AND OF THE NUMBER OF PIXELS(N)

		$N = 1 \times 10^5$		$N = 5 \times 10^5$	
	operation	$p = 10$	$p = 20$	$p = 10$	$p = 20$
VCA	$\mathbf{f}$ (lines 3-5)	0.09	0.57	0.09	0.57
	$\mathbf{v}$ (line 6)	535.16	445.68	2 677.62	2 193.38
	k (lines 7)	2.36	4.72	12.40	24.82
SUNSAL	$\mathbf{A}$ (line 2)	2 394.23	4 868.62	10 509.91	21 160.15
	$\mathbf{B}$ (line 3)	0.01	0.08	0.01	0.08
	$\mathbf{c}$ (line 4)	≈ 0.00	≈ 0.00	≈ 0.00	≈ 0.00
	$\mathbf{G}$ (line 5)	≈ 0.00	0.02	≈ 0.00	0.02
	$\mathbf{R}$ (line 8)	476.00	1 601.21	2 903.37	9 040.34
	$\mathbf{S}$ (line 9)	2 400.48	15 560.41	14 486.93	88 144.44
	$\mathbf{V}$ (lines 10)	169.72	572.17	1 034.20	3 210.11
	$\mathbf{U}$ (lines 11)	794.46	2 689.25	4 741.11	15 580.76
	$\mathbf{D}$ (lines 12)	337.27	1 127.14	2 044.35	6 414.02

1: $\widehat{\mathbf{M}} := \mathbf{0}_p$;
2: **for** $i := 1$ to p **do**
3: $\quad \mathbf{V} := \text{orth}\,(\widehat{\mathbf{M}})$ {Orthogonal vectors that spans $\widehat{\mathbf{M}}$ range}
4: $\quad \mathbf{P} := (\mathbf{I} - \mathbf{V}\mathbf{V}^T)$
5: $\quad \mathbf{f} :=$ generate a vector from span($\mathbf{P}$)
6: $\quad \mathbf{v} := \mathbf{f}^T\mathbf{Y}$;
7: $\quad k := \arg\max_{j=1,\ldots,N} |\mathbf{v}|$;
8: $\quad [\widehat{\mathbf{M}}]_{:,i} := [\mathbf{Y}]_{:,k}$;
9: **end for**

Fig. 2. Algorithm 1: VCA. Red and yellow lines denote operations to be computed in CPU and GPU, respectively.

1: choose $\mu > 0$, $\mathbf{U}_0$, and $\mathbf{D}_0$.
2: $\mathbf{A} := \mathbf{M}^T\mathbf{Y}$
3: $\mathbf{B} := \left(\mathbf{M}^T\mathbf{M} + \mu\mathbf{I}\right)^{-1}$
4: $\mathbf{c} := \mathbf{B}\mathbf{1}_p \left(\mathbf{1}_p^T\mathbf{B}\mathbf{1}_p\right)^{-1}$
5: $\mathbf{G} := \mathbf{B} - \mathbf{c}\mathbf{1}_p^T\mathbf{B}$
6: $k := 0$
7: **repeat**
8: $\quad \mathbf{R} := \mathbf{A} + \mu\left(\mathbf{U}_k + \mathbf{D}_k\right)$
9: $\quad \mathbf{S}_{k+1} := \mathbf{G}\mathbf{R} + \mathbf{c}\mathbf{1}_N^T$
10: $\quad \mathbf{V}_k := \mathbf{S}_{k+1} - \mathbf{D}_k$
11: $\quad \mathbf{U}_{k+1} := \max\{0, \mathbf{V}_k\}$
12: $\quad \mathbf{D}_{k+1} := \mathbf{D}_k - (\mathbf{S}_{k+1} - \mathbf{U}_{k+1})$
13: $\quad k := k + 1$
14: **until** stopping criterion is satisfied
15: $\widehat{\mathbf{S}} = \mathbf{S}_k$

Fig. 3. Algorithm 2: SUNSAL. Red and yellow lines denote operations to be computed in CPU and GPU, respectively.

The algorithms are constructed by chaining the so-called kernels which operate on entire streams and are executed by a multiprocessor, taking one or more streams as inputs and producing one or more streams as outputs.

The GPU architecture is organized to form a grid of blocks, where each block is composed by a group of threads that share data efficiently through the shared local memory and synchronize their execution for coordinating accesses to memory. There are different levels of memory in the GPU for the thread, block, and grid concepts. While the number of threads that can run in a single block is limited, the number of threads that can be concurrently executed is much larger since several blocks can be executed in parallel [16].

To implement VCA and SUNSAL on GPU, these methods were analyzed in order to determine the most consuming parts that can be parallelized. Table I presents the processing time of each step of both algorithms for a CPU (Intel core i7-2600) as a function of the number of endmembers ($p = \{10, 20\}$) and the number of pixels ($N = \{10^5, 5 \times 10^5\}$). It should be noted that these processing times only depend on the number of pixels (N) and the number of endmembers (p); hence, these can be generalized for any hyperspectral data set. It is clear that the projection of all pixels onto direction $\mathbf{f}$ inside the VCA loop (line 6), the calculation of $\mathbf{A}$ in SUNSAL (line 2), and the operations inside the SUNSAL loop (lines 8–12) are the most consuming parts of the method and they grow with p and N.

Assuming that VCA is applied after the projection of the data set onto the signal subspace [17], the computational complexity of VCA is $2p^2N$ floating-point operations, where the projection of each pixel of $\mathbf{Y}$ onto direction $\mathbf{f}_k$ can be done independently from the remaining pixels; thus, it can be parallelized. After the data set is transferred to the global memory of the GPU, using $4pN$ B on each iteration, the generation of the direction $\mathbf{f}_k$ is performed on the CPU and transferred to the constant memory of the device ($4p$ B). Then, a first kernel initially puts into execution as many threads as the number of pixel vectors of the image (N) divided into blocks of 32 threads, so that each thread is responsible for computing the dot product of $\mathbf{f}_k^{\mathrm{T}}$ by the pixel vector $\mathbf{Y}_i$, i.e., one element of vector $\mathbf{v}$. Then, the second kernel determines the index of the maximum absolute value of vector $\mathbf{v}$, indicating the position of the endmember signature on the data set. This task is performed using a binary reduction operation. These kernels correspond to the instructions in lines 6 and 7 of Algorithm 1 (yellow lines; Fig. 2). Fig. 4 presents an illustrative example of a single thread functioning. It is worth noting that to fully optimize the parallel algorithm, the size of $\mathbf{v}$ must be of the power of two (with zero padding if necessary).

The proposed implementation of SUNSAL follows the same rule, i.e., the most time-consuming operations are developed in a parallel fashion to be processed in the GPU. The operations outside the loop, namely, the inversion of the $p \times p$ matrix in line 3 of Algorithm 2 (see Fig. 3) are implemented in the CPU since it has a low computational cost (red lines in Algorithm 2). Inside the SUNSAL loop, the first kernel computes matrix $\mathbf{R}$ (see line 8 of Algorithm 2). This kernel launches as many threads as the number of elements present in $\mathbf{R}$, where each thread computes an element of $\mathbf{A} + \mu(\mathbf{U}_k + \mathbf{D}_k)$. The result is stored in the global memory. The second kernel computes the abundance estimates $\mathbf{S}$ on each iteration (line 9 of Algorithm 2) by first computing the product of matrices $\mathbf{G}$ and $\mathbf{R}$ and then adding matrix $\mathbf{c}\mathbf{1}_N^{\mathrm{T}}$. In order to minimize the number of global memory accesses, matrix $\mathbf{R}$ is partitioned into subblocks of 32×32 elements, which is the size of the block, and transferred to the shared memory. Each block uses a total of 8 KB of the

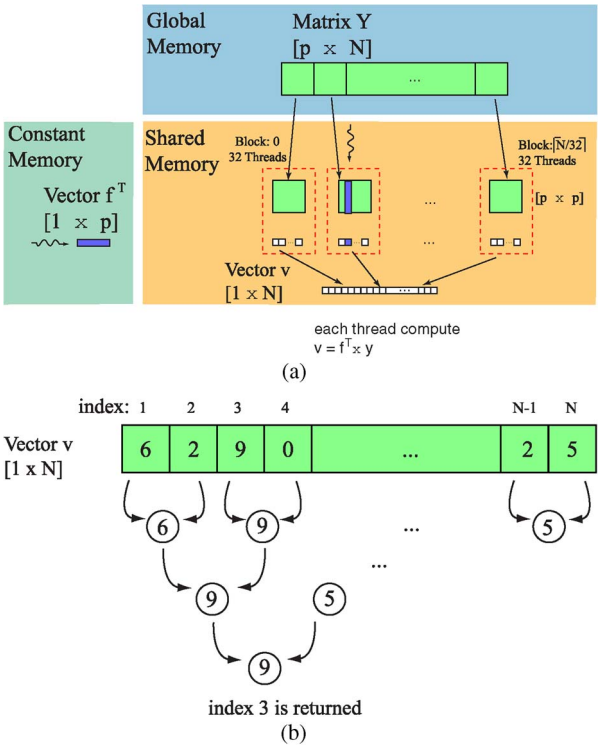

Fig. 4. Illustration of parallel VCA in the GPU. (a) Thread for computing $\mathbf{v} = \mathbf{f}^{\mathrm{T}}\mathbf{Y}$ and (b) thread for finding the index of the maximum absolute value of $\mathbf{v}$.

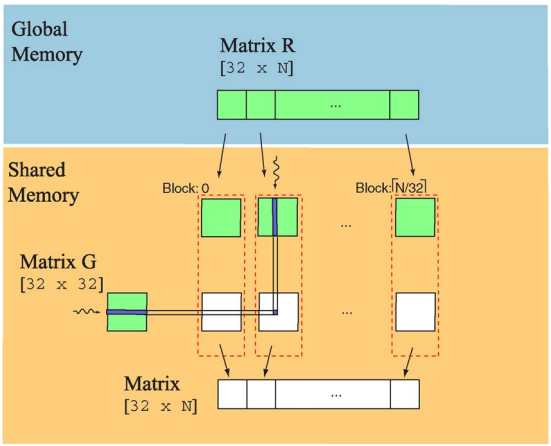

Fig. 5. Illustration of parallel SUNSAL in the GPU. Thread for computing one element of $\mathbf{RG}$.

shared memory. Fig. 5 illustrates this procedure, where one can see that each thread is responsible for computing each element of $\mathbf{S}$. The kernels for updating $\mathbf{V}$ and $\mathbf{D}$ follow the same strategy used on the first kernel, whereas $\mathbf{U}$ is updated by a kernel which analyzes if each element of $\mathbf{V}$ is negative. It should be noted that the matrices are stored in global memory which occupies around $28pN$ B.

IV. Performance Evaluation

In this section, we apply the sequential and parallel versions of the unmixing method based on VCA and SUNSAL for both the simulated data and the real data set of Cuprite collected by the AVIRIS sensor.

TABLE II
Characteristics of NVidia GPU Cards Used in the Tests

	GTX 590	GTX 680	C2050
Cores	1024	1536	448
Clock (GHz)	1215	1006	1150
Memory (GB)	3.0	2.0	3.0
Bandwidth (GB/s)	327.7	192.2	144

TABLE III
Processing Time and Data Transfer (in Seconds) for an Intel Core i7-2600 CPU and for a GTX590 GPU Card ($p = 30$ and $N = 10^5$)

	CPU	GTX 590
RAM → Global Mem.	-	0.014
VCA: f (lines 3 - 5)	0.002	
VCA loop (line 6 - 7)	1.479	0.008
RAM ← Global Mem.	-	0.001
SUNSAL: compute $\mathbf{A}$ (line 2)	7.230	0.423
SUNSAL (lines 3 - 5)	0.036	
RAM → Global Mem.	-	0.221
SUNSAL loop (lines 8 - 12)	57.39	0.624
RAM ← Global Mem.	-	0.028
Total Time	65.389	1.607
Speedup (CPU time / GTX590 time)	-	48.12

In order to evaluate the performance of the proposed method in terms of processing time, the sequential version was implemented in C programming language running on a computer platform equipped with a quad-core Intel i7-2600 CPU, 3.4-GHz clock speed, 16-GB memory, and 1 and 8 MB of L2 and L3 cache memory, respectively. The parallel version was implemented in OpenCL programming language for three different GPU cards from NVidia. Table II presents a summary of the characteristics of the three cards. The following subsections will present the performance results in terms of acceleration factors or speedups.

A. Evaluation With Simulated Data

Several synthetic scenes are created with different numbers of pixels and endmembers. Each pixel is generated according to (1), where spectral signatures are selected from the USGS digital spectral library containing 224 spectral bands covering wavelengths from 0.38 to 2.5 μm with a spectral resolution of 10 nm. The abundance fractions are generated according to a Dirichlet distribution which enforces positivity and full additivity constraints (see [18] for details).

Table III illustrates the data transfer from/to device time, processing time, and speedup factors for a data set composed of 10^5 pixels with 30 endmembers ($p = 30$). The processing time on the GPU is lower than the CPU time due to the parallel processing implementation.

Fig. 6 shows the speedup of the parallel version (with regard to the sequential version) for three different GPU cards as a function of the number of endmembers and for $N = 5 \times 10^5$. For illustration purposes, we also present the results for the method running on all CPU cores. Herein, the method implemented in OpenCL is compiled for the quad-core Intel processor. The speedup of GTX680 is higher than 100, for $p = 30$, which is quite remarkable taking into account the fact that the sequential version has been carefully optimized. As expected, the speedup grows with the number of endmembers.

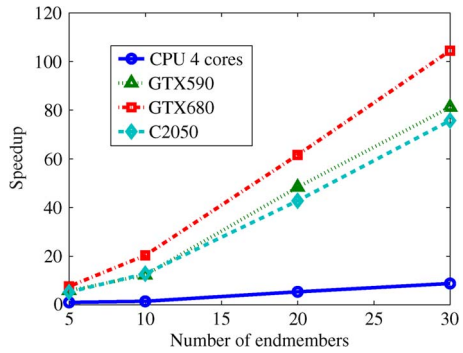

Fig. 6. Speedup of parallel version as a function of the number of endmembers (p) for $N = 5 \times 10^5$.

TABLE IV
PROCESSING TIME (IN SECONDS) FOR THE CUPRITE DATA SET

	Total time	VCA	SUNSAL	Speedup
CPU	98.270	0.400	97.870	-
CPU (4 cores)	19.537	0.296	19.241	5.029
GTX590	1.169	0.021	1.148	84.06
GTX680	0.975	0.021	0.954	100.79
Tesla C2050	1.367	0.022	1.345	71.88

Note that the GTX680 GPU card, which has more cores, has the best performance, and the quad-core CPU has a speedup smaller than 8, achieving always the worst result.

B. Evaluation With Real Data

In this section, the proposed method is applied to real hyperspectral data collected by the AVIRIS sensor. A subset of the Cuprite data set containing 350×350 pixels with 187 spectral bands (noisy and water absorption bands were removed) is considered. This site has been extensively used for the evaluation of spectral unmixing applications, where the presence of 14 pure materials has been determined. The geological properties of this site have been extensively reported [19].

Table IV shows the processing times (in seconds) for the CPU sequential versions of VCA and SUNSAL and also for the parallel versions using three different GPUs. One can note that the best speedup is higher than 100 times, achieved on the GTX680 card. These results are in agreement with the ones for the simulated scenarios.

Finally, it is worth mentioning that as the AVIRIS sensor is able to collect 512 hyperspectral pixels in 8.3 ms [20], a 350×350 subimage takes nearly 2 s. Consequently, the proposed parallel method using GPUs is suitable for real-time hyperspectral unmixing systems.

V. CONCLUSION

In this letter, a new parallel implementation of VCA and SUNSAL on GPUs has been proposed. The significant speedup reported in the experiments will bridge the gap toward real-time spectral unmixing of hyperspectral data sets, which is a highly sought-after requirement for many remote sensing applications. The developed methods were designed using the multicore/multiplatform OpenCL programming language in order to easily migrate the application to new and powerful hardware platforms, such as multi-GPU systems, clusters of GPUs, and field-programmable gate array board systems.

ACKNOWLEDGMENT

Part of this work was conducted within the Labex Centre International de Mathématiques et Informatique de Toulouse during the visits of J. Bioucas-Dias at the University of Toulouse.

REFERENCES

[1] J. Bioucas-Dias, A. Plaza, N. Dobigeon, M. Parente, Q. Du, P. Gader, and J. Chanussot, "Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 5, no. 2, pp. 354–379, Apr. 2012.

[2] 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, May 2011.

[3] J. M. P. Nascimento and J. M. Bioucas-Dias, "Does independent component analysis play a role in unmixing hyperspectral data?" *IEEE Trans. Geosci. Remote Sens.*, vol. 43, no. 1, pp. 175–187, Jan. 2005.

[4] N. Keshava, J. Kerekes, D. Manolakis, and G. Shaw, "An algorithm taxonomy for hyperspectral unmixing," in *Proc. SPIE AeroSense Conf. Algorithms Multispectr. Hyperspectr. Imagery VI*, 2000, vol. 4049, pp. 42–63.

[5] J. M. P. Nascimento and J. M. Bioucas-Dias, "Vertex component analysis: A fast algorithm to unmix hyperspectral data," *IEEE Trans. Geosci. Remote Sens.*, vol. 43, no. 4, pp. 898–910, Apr. 2005.

[6] A. Plaza, P. Martinez, R. Perez, and J. Plaza, "Spatial/spectral endmember extraction by multidimensional morphological operations," *IEEE Trans. Geosci. Remote Sens.*, vol. 40, no. 9, pp. 2025–2041, Sep. 2002.

[7] J. Boardman, "Automating spectral unmixing of AVIRIS data using convex geometry concepts," in *Summaries 4th Annu. JPL Airborne Geosci. Workshop, JPL Pub. 93-26, AVIRIS Workshop.*, 1993, vol. 1, pp. 11–14.

[8] M. E. Winter, "N-FINDR: An algorithm for fast autonomous spectral end-member determination in hyperspectral data," in *Proc. SPIE Conf. Imaging Spectr. V*, 1999, vol. 3753, pp. 266–275.

[9] T.-H. Chan, W.-K. Ma, A. Ambikapathi, and C.-Y. Chi, "A simplex volume maximization framework for hyperspectral endmember extraction," *IEEE Trans. Geosci. Remote Sens.*, vol. 49, no. 11, pp. 4177–4193, Nov. 2011.

[10] N. Gillis and S. Vavasis, Fast and Robust Recursive Algorithms for Separable Nonnegative Matrix Factorization 2012, arXiv preprint arXiv:1208.1237.

[11] D. Heinz and C.-I. Chang, "Fully constrained least squares linear spectral mixture analysis method for material quantification in hyperspectral imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 39, no. 3, pp. 529–545, Mar. 2001.

[12] J. Bioucas-Dias and M. Figueiredo, "Alternating direction algorithms for constrained sparse regression: Application to hyperspectral unmixing," in *Proc. 2nd WHISPERS*, 2010, pp. 1–4.

[13] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, Jan. 2011.

[14] A. Barberis, G. Danese, F. Leporati, A. Plaza, and E. Torti, "Real-time implementation of the vertex component analysis algorithm on GPUs," *IEEE Geosci. Remote Sensing Lett.*, vol. 10, no. 2, pp. 251–255, Mar. 2013.

[15] S. Sanchez, G. Martin, A. Plaza, and C.-I. Chang, "GPU implementation of fully constrained linear spectral unmixing for remotely sensed hyperspectral data exploitation," in *Proc. SPIE 7810, Satellite Data Compression, Commun. Process. VI*, 2010, vol. 7810, pp. 78100G-11–78100G-11.

[16] T. Han and T. Abdelrahman, "hICUDA: High-level GPGPU programming," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 1, pp. 78–90, Jan. 2011.

[17] J. M. Bioucas-Dias and J. M. P. Nascimento, "Hyperspectral subspace identification," *IEEE Trans. Geosci. Remote Sens.*, vol. 46, no. 8, pp. 2435–2445, Aug. 2008.

[18] J. M. P. Nascimento and J. M. Bioucas-Dias, "Hyperspectral unmixing based on mixtures of Dirichlet components," *IEEE Trans. Geosci. Remote Sens.*, vol. 50, no. 3, pp. 863–878, Mar. 2012.

[19] G. Swayze, R. Clark, S. Sutley, and A. Gallagher, "Ground-truthing AVIRIS mineral mapping at Cuprite, Nevada," in *Summaries 3rd Annu. JPL Airborne Geosci. Workshop*, 1992, pp. 47–49.

[20] S. Lopez, P. Horstrand, G. Callico, J. Lopez, and R. Sarmiento, "A novel architecture for hyperspectral endmember extraction by means of the Modified Vertex Component Analysis (MVCA) algorithm," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 5, no. 6, pp. 1837–1848, Dec. 2012.