GPU Implementation of Spatial–Spectral Preprocessing for Hyperspectral Unmixing

Luis Ignacio Jiménez, Gabriel Martín, Sergio Sánchez, Carlos García, Sergio Bernabé, Javier Plaza, *Senior Member, IEEE*, and Antonio Plaza, *Fellow, IEEE*

***Abstract*—Spectral unmixing pursues the identification of spectrally pure constituents, called *endmembers*, and their corresponding *abundances* in each pixel of a hyperspectral image. Most unmixing techniques have focused on the exploitation of spectral information alone. Recently, some techniques have been developed to take advantage of the complementary information provided by the spatial correlation of the pixels in the image. Computational complexity represents a major problem in these spatial–spectral techniques, as hyperspectral images contain very rich information in both the spatial and spectral domains. In this letter, we develop a computationally efficient implementation of a spatial–spectral processing algorithm that has been successfully applied prior to the spectral unmixing of the hyperspectral data. Our implementation has been optimized for the commodity graphics processing units (GPUs) and is evaluated (using both synthetic and real data) using different GPU architectures. Significant speedups can be achieved when processing hyperspectral images of different sizes. This allows for the inclusion of the proposed parallel preprocessing module in a full hyperspectral unmixing chain able to operate in real time.**

***Index Terms*—Graphics processing units (GPUs), hyperspectral unmixing, spatial–spectral preprocessing (SSPP).**

I. Introduction

The wealth of spectral information provided by imaging spectrometers has promoted the application of hyperspectral imaging techniques in many different areas of interest [1]. In hyperspectral unmixing, endmember extraction is the process of collecting pure signature spectra of the materials present in a remotely sensed hyperspectral scene. These pure signatures are then used to decompose the scene into a set of so-called abundance fractions, representing the coverage of each endmember in each image pixel.

Several algorithms have been developed for automatic or semi-automatic identification of endmembers over the last decade [2]. A majority of the algorithms have been developed under the pure pixel assumption, i.e., they assume that the remotely sensed data contain one pure observation for each different material in the scene [3]. These algorithms often rely exclusively on the exploitation of spectral information in order to select the final set of endmembers. However, spatial information can greatly assist in the unmixing task by considering local structures latent in the data [4].

In order to also include the spatial information, several techniques have been proposed in the literature, such as the automatic morphological endmember extraction [5] or the spatial–spectral endmember extraction [6]. Furthermore, several spatial preprocessing algorithms have been developed that can be applied prior to any spectral-based endmember extraction technique. Techniques include the spatial preprocessing (SPP) [7], region-based SPP [8], and spatial–spectral preprocessing (SSPP) [9]. The goal of these preprocessing methods is to guide the search for endmembers using not only spectral but also spatial information, which can greatly assist in the selection of more spatially representative endmembers without the need to modify the endmember identification algorithm (the preprocessing can be applied as an optional step). Such SPP adds some extra computational cost to the full spectral unmixing chain. As a result, the development of efficient implementations for SPP techniques has become an important goal.

In this letter, we present a new parallel implementation of the SSPP algorithm, which has been shown to be one of the most successful SPP techniques available in the literature [9]. Our implementation has been developed for commodity graphics processing units (GPUs) [10] and tested on several GPU architectures. Synthetic scenes are used to validate the efficacy of the implementation, whereas real hyperspectral data are used to evaluate a full unmixing chain that includes our efficient preprocessing module. The results indicate that significant speedups can be achieved, allowing us to embed the SSPP into a full unmixing chain that performs in real time after including the SPP module.

The remainder of this letter is organized as follows. Section II enumerates and describes the different steps of the SSPP method. Section III describes the proposed parallel implementation for GPUs. Section IV describes the experiments conducted using synthetic data, as well as the real data experiments intended to evaluate the acceleration achieved by our parallel implementation in the context of a full hyperspectral unmixing chain. Section V concludes this letter with some remarks and hints at plausible future research lines.

II. SSPP

This section briefly outlines the SSPP algorithm in [9]. As shown in the flowchart given in Fig. 1, the SSPP method consists of the following steps.

1) *Multiscale Gaussian filtering*. This step takes as input the original hyperspectral image $\mathbf{Y}^{I\times J\times B}$, where I is the

Manuscript received February 23, 2016; revised June 17, 2016; accepted August 8, 2016. Date of publication September 8, 2016; date of current version October 12, 2016. This work was supported by Junta de Extremadura (decreto 297/2014, ayudas para la realización de actividades de investigación y desarrollo tecnológico, de divulgación y de transferencia de conocimiento por los Grupos de Investigación de Extremadura, Ref. GR15005).

L. I. Jiménez, S. Sánchez, J. Plaza, and A. Plaza are with the Hyperspectral Computing Laboratory, University of Extremadura, 10071 Cáceres, Spain (e-mail: luijimenez@unex.es; sersanmar@unex.es; jplaza@unex.es; aplaza@unex.es).

G. Martín is with Instituto de Telecomunicações, 1049-001 Lisbon, Portugal (e-mail: gabriel.hernandez@lx.it.pt).

C. García and S. Bernabé are with Complutense University of Madrid, 28040 Madrid, Spain.

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.2016.2602227

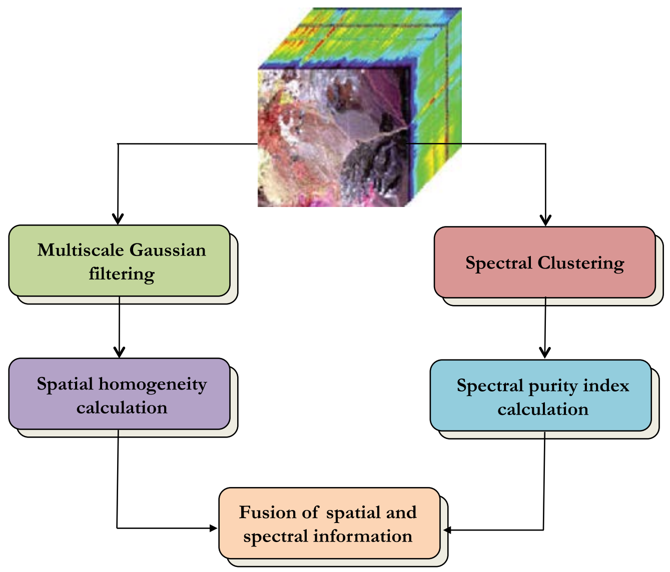
Multiscale Gaussian filtering
Spectral Clustering
Spatial homogeneity calculation
Spectral purity index calculation
Fusion of spatial and spectral information

Fig. 1. Block diagram illustrating the SSPP method.

number of rows, J is the number of columns, and B is the number of bands, and returns a filtered version of the image. To perform this step, we first apply Gaussian filtering to each of the B spectral bands of the hyperspectral image. According to the results in [9], we use $\sigma = 1.2$ for defining the Gaussian filter. This results in a filtered version $\mathbf{Y}_F$ of the original hyperspectral image. Let us denote by $\mathbf{y}(i,j) = [y_1(i,j), y_2(i,j), \ldots, y_B(i,j)]$ the B-dimensional pixel vector at spatial coordinates (i,j) of the hyperspectral image $\mathbf{Y}$, which can now be defined as a set $\mathbf{Y} = \{\mathbf{y}(i,j)\}_{i \in 1,\ldots,r}$. Equation (1) shows the pixel-level operation that we perform for each kth spectral band of the hyperspectral image, with $1 \leq k \leq B$, i.e.,

$$F_k\left[\mathbf{y}(i,j)\right] = \sum_{i'=1}^{r}\sum_{j'=1}^{c} G(i-i', j-j') \cdot y_k(i',j')$$
$$\text{with } G(i',j') = \frac{1}{2\pi\sigma^2} e^{-\frac{i'^2 + j'^2}{2\sigma^2}}. \quad (1)$$

2) *Spatial homogeneity calculation*. This step takes as input the filtered hyperspectral image obtained in the previous step and produces a spatial homogeneity index for each pixel in the original image $\mathbf{Y}$. To perform this step, we first calculate the root-mean-square error (RMSE) [11] between the original hyperspectral image and the filtered image. Equation (2) indicates the operation to calculate the RMSE between the pixel $\mathbf{y}(i,j)$ in the original image and the pixel at the same spatial coordinates, i.e., $\mathbf{y}_F(i,j)$, in the filtered image

$$\mathrm{RMSE}[\mathbf{y}(i,j), \mathbf{y}_F(i,j)] = \left(\frac{1}{B}\sum_{k=1}^{B}\left(y_k(i,j) - y_{F_k}(i,j)\right)^2\right)^{\frac{1}{2}}. \quad (2)$$

The lower the RMSE score, the higher the similarity between the pixels in the original image and its neighbors. Quite the opposite, the higher the RMSE, the lower the similarity of the pixel in the original image with regard to its neighbors. As a result, the RMSE in (2) can be used as a spatial homogeneity index for each pixel $\mathbf{y}(i,j)$ in the hyperspectral image $\mathbf{Y}$.

3) *Spectral purity index calculation*. For this step, we first use the principal component analysis (PCA) [12] to reduce the dimensionality of the hyperspectral image, retaining the first p principal components (PCs) containing most of the variance in the data. Then, we use the first PCs as the skewers for which we identify the pixels with maxima and minima projection values, following a procedure similar to the one adopted by the pixel purity index algorithm in [13]. The pixels with maxima and minima projection values are assigned a weight of 1. The weight of the mean value between the maxima and minima projection value is 0. A threshold value, set empirically to $\theta = 0.7$ in this work [9], is also applied so that the weights lower than this threshold are assigned the value 0. Finally, the spectral purity is calculated as the sum of all the weights over the first p PCs.

4) *Spectral clustering*. At this point, we perform a spectral-based unsupervised clustering of the original hyperspectral image. This step, which is separately applied from the previous steps, uses the K-means algorithm [12] in order to identify p clusters in the hyperspectral image.

5) *Fusion of spatial and spectral information*. This step takes as input the spatial homogeneity index calculated in the second step and the clusters calculated in the fourth step and returns a subset of candidate pixels in the original hyperspectral image which will be used for endmember identification purposes. For each cluster, a subset of spatially homogeneous and spectrally pure pixels is selected. To do so, pixels in each cluster are ranked according to the increasing values of their spatial homogeneity and spectral purity.

Finally, an endmember extraction algorithm can be applied to the pixels retained after the aforementioned procedure. The outcome of the process is a set of p endmembers and their corresponding fractional abundance maps (one per endmember).

III. GPU Implementation of SSPP

The parallel implementation of the SPP has been developed using the NVidia Compute Unified Device Architecture (CUDA)[1] version 6.5.12 for Unix platforms, which uses the GNU Compiler Collection (GCC) compiler version 4.8.2. Our implementation is in fact a hybrid CPU/GPU one, in the sense that the tasks are allocated to either the CPU or the GPU depending on their nature, with overlapping CPU/GPU computations, as indicated in Fig. 2. This strategy aims at optimizing the performance as much as possible. As shown in Fig. 2, there are some synchronization points due to inherent data dependencies. The most computationally expensive task executed in the CPU is the singular value decomposition (SVD) calculation required for the PCA reduction, whereas the reduced image calculation is the most computationally expensive part in the GPU. In order to obtain the reduced image in the GPU, the SVD execution in the CPU must have finished, thus a first synchronization point is required. A second synchronization point is required at the end, just before the fusion of spectral and spatial information takes place in the CPU (see Fig. 2). This is due to the fact that the transfer of the required data to the GPU would penalize the performance beyond the execution of this step in the CPU. In our

[1]https://developer.nvidia.com/cuda-zone

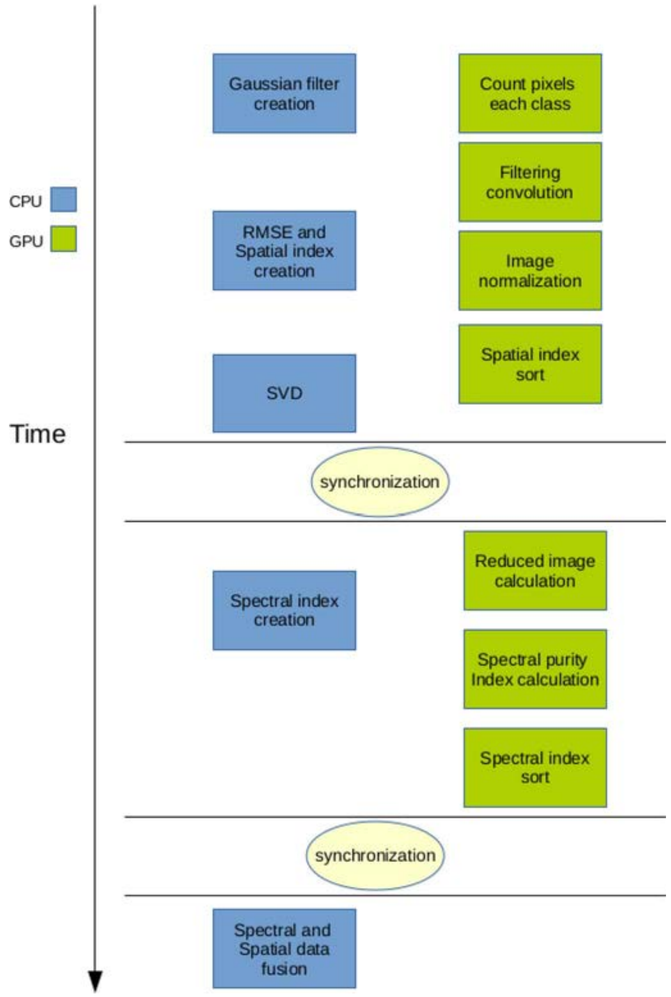

Fig. 2. CPU/GPU hybrid implementation of SSPP.

implementation, we also use some predefined cuBLAS[2] functions for efficient matrix operations, which represent around 55% of the total computation time in the GPU, and a simple kernel that counts the number of pixels that belong to each cluster (called `countingPixelsClusters`). In addition, we use the following CUDA kernels.

1) Kernels `convolutionRowSymmGPU` and `convolutionColSymmGPU` perform the multiscale Gaussian filtering step. They accomplish the symmetric Gaussian filtering using the convolution of two 1-D filters implemented in each kernel. Here, the number of processing threads in the GPU is set to 128 empirically after analyzing the optimal configuration in terms of the number of blocks in the GPU (a block is defined as a group of threads that share a local cache memory). We have empirically tested that, if the number of threads is set to a smaller value, the number of blocks is increased, and this results in worse computational times (the number of blocks is equal to the number of pixels divided by the number of threads). Both kernels allocate the filter (with size $s = 15$) in the shared memory. Gaussian filtering is applied to each of the spectral bands of the image; thus, each of the two kernels needs to be executed B times.

[2]https://developer.nvidia.com/cublas

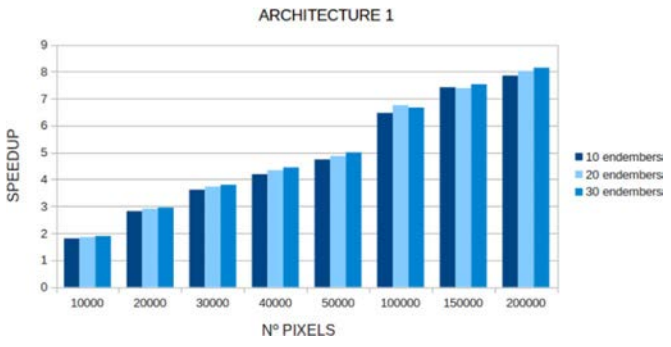

Fig. 3. Speedup of the GPU version of the SSPP algorithm applied to synthetic images with different sizes and numbers of endmembers (10, 20, and 30) on Architecture 1. The values reported are the average after ten Monte Carlo runs.

2) Kernel `AvgXCUDA` calculates the normalized image obtained by subtracting the average of all pixels in the scene to each pixel in the original image, using a reduction operation. This kernel is part of the PCA operation required to calculate the spectral purity index. The number of blocks is set to the number of bands, i.e., B; and the number of threads is set to the maximum value allowed by the considered GPU architecture.

3) We have also developed a kernel called `BitonicSort` that replaces the Quicksort algorithm (which is commonly executed in the CPU). This kernel is used to accelerate the sorting processes. Specifically, we use the bitonic algorithm for both spectral- and spatial-based sorting operations. In order to increase the overlapping factor between the GPU and the CPU, the calls to this kernel are always performed from different blocks. Here, the number of threads is empirically set to 256 and the number of blocks is calculated, according to

$$\text{nblocks} = \exp\left(\log_2\left\lfloor\frac{\text{npixels}}{\text{nthreads}}\right\rfloor\right). \quad (3)$$

4) After executing the PCA reduction step, the rest of the spectral purity index calculation is implemented through kernels, i.e., `maxminbands` and `weights`. The first kernel calculates the maxima and minima projections in the PCA domain, using two reduction operations in each case. The second kernel obtains the weights of each minima and maxima projection on each PC. The grid dimension of the `maxminbands` kernel (where the grid is defined as a logical structure that contains a number of blocks) is set to the number of PCs, whereas the grid dimension in the `weights` kernel is obtained by dividing the number of pixels by the block size, which, in both cases, is set to the maximum value by the considered GPU architecture.

IV. Experimental Results

The parallel version of SSPP exactly achieves the same results as the serial implementation in [9]. Therefore, we focus on the analysis of the parallel performance of the proposed implementation. First, a set of experiments were conducted using a collection of 24 synthetic hyperspectral images simulated with different sizes and numbers of endmembers. In this case, we focus on analyzing the performance of the SSPP

TABLE I
MEAN EXECUTION TIMES (IN SECONDS) AND SPEEDUPS (IN THE PARENTHESES) AFTER EXECUTING THE PARALLEL SSPP (MULTICPU AND GPU VERSIONS) USING SYNTHETIC HYPERSPECTRAL IMAGES. THE VALUES REPORTED ARE THE AVERAGE AFTER TEN MONTE CARLO RUNS

	Architecture 1					
	10 endmembers		20 endmembers		30 endmembers	
Image size	MultiCPU	GPU	MultiCPU	GPU	MultiCPU	GPU
100 × 100	0.243 (1.723)	0.231 (1.809)	0.150 (2.781)	0.225 (1.856)	0.148 (2.847)	0.222 (1.899)
100 × 200	0.249 (3.044)	0.269 (2.82)	0.304 (2.502)	0.261 (2.91)	0.275 (2.791)	0.260 (2.952)
100 × 300	0.345 (3.162)	0.302 (3.615)	0.354 (3.119)	0.296 (3.729)	0.442 (2.508)	0.292 (3.797)
100 × 400	0.444 (3.239)	0.343 (4.192)	0.535 (2.709)	0.334 (4.34)	0.456 (3.206)	0.329 (4.444)
100 × 500	0.586 (3.039)	0.376 (4.737)	0.541 (3.346)	0.372 (4.868)	0.589 (3.111)	0.366 (5.003)
500 × 200	1.072 (3.271)	0.542 (6.467)	1.061 (3.393)	0.533 (6.752)	1.122 (3.196)	0.538 (6.665)
500 × 300	1.506 (3.478)	0.706 (7.417)	1.639 (3.213)	0.713 (7.387)	1.516 (3.488)	0.702 (7.533)
500 × 400	1.917 (3.563)	0.870 (7.85)	2.101 (3.355)	0.878 (8.028)	2.149 (3.294)	0.869 (8.145)

TABLE II
MEAN EXECUTION TIMES (IN SECONDS) AND SPEEDUPS (IN THE PARENTHESES) AFTER EXECUTING A FULL GPU UNMIXING CHAIN USING REAL HYPERSPECTRAL DATA. THE VALUES REPORTED ARE THE AVERAGE AFTER TEN MONTE CARLO RUNS

	Architecture 1		**Architecture 2**		**Architecture 3**	
Algorithm	Cuprite	WTC	Cuprite	WTC	Cuprite	WTC
VD	0.393 (1.831)	0.702 (3.226)	0.260 (1.745)	0.573 (2.736)	0.450 (12.765)	0.949 (22.877)
K-MEANS	1.035 (3.274)	3.512 (5.946)	0.513 (4.998)	1.502 (10.119)	1.553 (2.524)	8.420 (2.681)
SSPP	0.485 (7.110)	1.341 (8.212)	0.352 (6.113)	0.990 (7.126)	0.670 (13.234)	1.744 (18.70)
N-FINDR	0.035 (2.589)	0.265 (9.748)	0.021 (0.968)	0.175 (7.550)	0.028 (1.250)	0.160 (16.481)
LSU	0.050 (3.371)	0.164 (3.489)	0.045 (2.108)	0.140 (2.633)	0.067 (9.160)	0.210 (14.152)
TOTAL	1.998 (3.912)	5.983 (6.236)	1.191 (4.437)	3.381 (7.548)	2.768 (6.927)	11.483 (7.104)

algorithm alone. On the other hand, a second set of experiments is designed to evaluate a full parallel unmixing chain, including the newly designed parallel SPP module. The full chain is applied to the two different hyperspectral data sets. Three different architectures have been considered.

1) Architecture 1: A desktop computer (Intel core i7 920 CPU at 2.67 GHz and 6 GB of RAM) with an NVidia GTX 580 GPU equipped with 512 processor cores operating at 1.54 GHz and 1536 MB of RAM memory.
2) Architecture 2: A quad-core desktop computer (Intel i-7 4790 at 3.6 GHz and 32 GB of RAM) with a NVidia GeForce GTX 980 GPU equipped with 16 multiprocessors containing 128 CUDA cores each (2048 CUDA cores in total) at 1.33 GHz and 4 GB of RAM memory.
3) Architecture 3: A compute cluster[3] with 44 Nvidia TESLA S2070 GPU nodes (2 M2075 per node), each with an Intel Xeon CPU E5645 at 2.40 GHz and a total of 24 GB of RAM, divided in 12 modules of 2 GB each.

In all cases, the serial algorithm was executed in one of the available cores, and the parallel times were measured using all the cores in each GPU platform. The speedups are calculated between each CPU/GPU pair. For each experiment, ten Monte Carlo runs were performed (the times were very similar, with differences on the order of a few milliseconds only).

A. Synthetic Data Experiments

A collection of 24 synthetic hyperspectral images were first used for validation. The procedure for constructing the images is fully described in [3] and [9]. The images contain clusters of (10, 20, 30) endmembers randomly selected from the U.S.

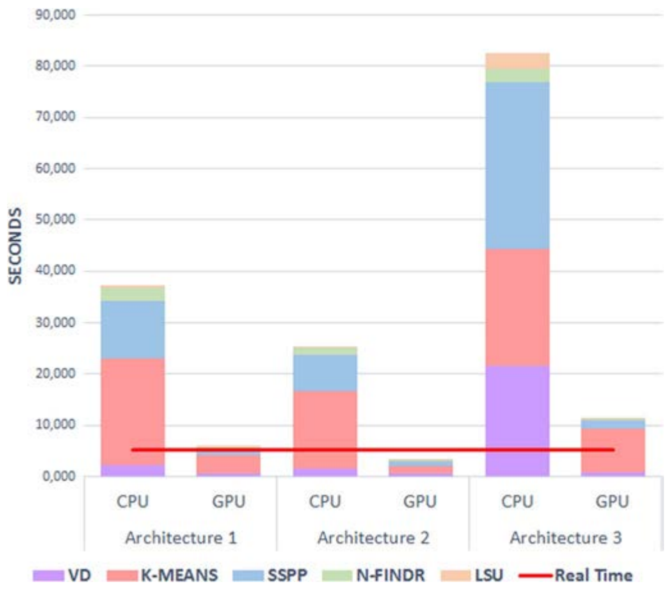

Fig. 4. Execution time for each step of the full unmixing chain for the WTC data set. The red line shows the real-time performance threshold.

Geological Survey library,[4] and comprise 221 narrow spectral bands between 0.4 and 2.5 μm. Fig. 3 shows the speedups obtained by the GPU implementation of SSPP on Architecture 1 for synthetic images with different sizes and numbers of endmembers. For comparative purposes, Table I also includes the execution times and speedups obtained by a multiple threads version (MultiCPU) implemented using OpenMP and the Intel ICC compiler version 14.0.2 with $-O3$ and $-$restrict flags. Table I reveals better performance of both the GPU and

[3]http://www.ceta-ciemat.es/

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

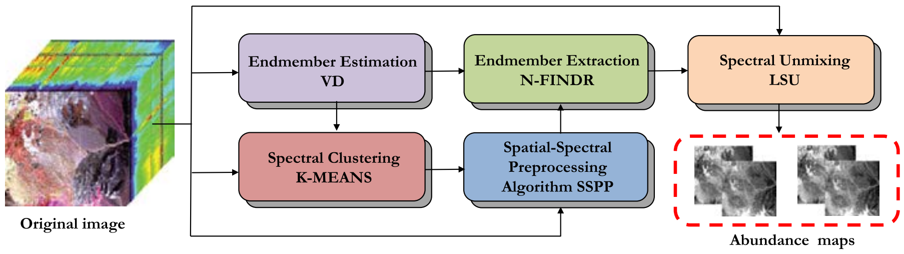

Fig. 5. Fully parallel hyperspectral unmixing chain.

MultiCPU versions as the number of endmember increases. In the GPU case, this is a consequence of the implementation strategy considered for `maxminbands` and `weights` kernels. Last but not least, the linear trend of the results suggests that real data sets with larger sizes may allow for the inclusion of SSPP in a full unmixing chain able to operate in real time.

B. Real Data Experiments

We have used two real hyperspectral data sets. The first one was collected by the Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS) instrument, which is operated by the NASA's Jet Propulsion Laboratory, over the Cuprite mining district in Nevada in the summer of 1997 (these data are available online).[5] The portion used in the experiments corresponds to a 350 $\times$ 350 pixel subset with 188 spectral bands in the range from 400 to 2500 μm, and a total size of 50 MB (several bands were removed due to water absorption and low signal to noise ratio in those bands). The second hyperspectral scene was also collected by AVIRIS over the World Trade Center (WTC) area in New York City on September 16, 2001 just five days after the terrorist attacks that collapsed the two main towers and other buildings in the WTC complex.[6] The full data set selected for experiments consists of 614 $\times$ 512 pixels, 224 spectral bands, and a total size of $\approx$140 MB.

The unmixing chain used in these experiments is composed of five different parallel steps (see Fig. 5). The GPU implementation of the different parts of the chain is described in [14]. The estimated number of endmembers is $p = 25$ for the Cuprite scene and $p = 31$ for the WTC scene. Table II shows the time for each step of the parallel unmixing chain (with our GPU version of SSPP embedded) and the obtained speedups in three different GPU architectures. Architectures 1 and 2 (based on NVidia GTX devices) obtain better performance than Architecture 3 (based on NVidia TESLA GPU devices). This is because the NVidia TESLA includes error checking and correction that guarantees more stable results at the expense of a slightly reduced performance. Fig. 4 shows the execution times of each step of the unmixing chain for the WTC data set. The time taken by data transfers between the CPU and the GPU is included in the execution times reported in Fig. 4. Such overhead represents 16.63%, 21.94%, and 9.4% of the total execution time for Architectures 1, 2, and 3, respectively. In the case of AVIRIS (a pushbroom instrument), the cross-track line scan time is quite fast (8.3 ms to collect 512 full pixel vectors). For real-time performance, the WTC image (512 $\times$ 614 pixel vectors) needs to be processed in approximately 5.2 s, which results from a data collection rate of approximately 27 MB/s. As shown in Table II, the execution of the SSPP algorithm is always below this threshold. In addition, the full unmixing chain is below the threshold in the case of Architecture 2.

[5]http://aviris.jpl.nasa.gov
[6]http://speclab.cr.usgs.gov/wtc/

V. Conclusions and Future Lines

In this letter, we have developed a new GPU implementation of the SSPP algorithm. The obtained results indicate that it is possible to achieve significant speedups by overlapping the execution of the algorithm in the CPU/GPU. The parallel SSPP has been embedded into a full real-time unmixing chain. Future work will focus on improving this implementation by studying different clustering and endmember extraction algorithms.

References

[1] A. Goetz, G. Vane, J. Solomon, and B. Rock, "Imaging spectrometry for earth remote sensing," *Science*, vol. 228, no. 4704, pp. 1147–1153, 1985.
[2] J. M. Bioucas-Dias *et al.*, "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, Feb. 2012.
[3] J. Plaza, E. M. T. Hendrix, I. Garcia, G. Martin, and A. Plaza, "On endmember identification in hyperspectral images without pure pixels: A comparison of algorithms," *J. Math. Imag. Vis.*, vol. 42, no. 2/3, pp. 163–175, 2012.
[4] F. Zhu, Y. Wang, S. Xiang, B. Fan, and C. Pan, "Structured sparse method for hyperspectral unmixing," *ISPRS J. Photogramm. Remote Sens.*, vol. 88, pp. 101–118, 2014.
[5] 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.
[6] D. M. Rogge, B. Rivard, J. Zhang, A. Sanchez, J. Harris, and J. Feng, "Integration of spatial-spectral information for the improved extraction of endmembers," *Remote Sens. Environ.*, vol. 110, no. 3, pp. 287–303, 2007.
[7] M. Zortea and A. Plaza, "Spatial preprocessing for endmember extraction," *IEEE Trans. Geosci. Remote Sens.*, vol. 47, no. 8, pp. 2679–2693, Aug. 2009.
[8] G. Martin and A. Plaza, "Region-based spatial preprocessing for endmember extraction and spectral unmixing," *IEEE Geosci. Remote Sens. Lett.*, vol. 8, no. 4, pp. 745–749, Apr. 2011.
[9] G. Martin and A. Plaza, "Spatial-spectral preprocessing prior to endmember identification and unmixing of remotely sensed hyperspectral data," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 5, no. 2, pp. 380–395, Feb. 2012.
[10] J. M. P. Nascimento, J. M. Bioucas-Dias, J. M. Rodriguez Alves, V. Silva, and A. Plaza, "Parallel hyperspectral unmixing on GPUs," *IEEE Geosci. Remote Sens. Lett.*, vol. 11, no. 3, pp. 666–670, Mar. 2013.
[11] N. Keshava and J. F. Mustard, "Spectral unmixing," *IEEE Signal Process. Mag.*, vol. 19, no. 1, pp. 44–57, Jan. 2002.
[12] J. A. Richards and X. Jia, *Remote Sensing Digital Image Analysis*. Berlin, Germany: Springer-Verlag, 1999.
[13] J. W. Boardman, F. A. Kruse, and R. O. Green, "Mapping target signatures via partial unmixing of AVIRIS data," in *Proc. JPL Airborne Earth Sci. Workshop*, 1995, pp. 23–26.
[14] S. Sánchez, R. Ramalho, L. Sousa, and A. Plaza, "Real-time implementation of remotely sensed hyperspectral image unmixing on GPUs," *J. Real-Time Image Process.*, vol. 10, no. 3, pp. 469–483, 2015.